Natural Language Processing

AI51701/CSE71001

Lecture 20

11/28/2023

Instructor: Taehwan Kim

Announcement

- Reminder: no class on 12/5 & 12/7
  - The next lecture is our last lecture except the final project presentations on 12/12 and 12/4

Code Generation

(adapted from Gabriel Poesia's slides in Stanford CS224n)

Background: Program Synthesis

Program Synthesis

- Major long-standing challenge of AI: programs that write programs!
- Program synthesizer: program that takes a specification and outputs a program that satisfies it
- What specification?
  - A logical formula
  - Another equivalent program (e.g., a slower one)
  - Input/output examples
  - **Natural language description**

Program Synthesis from Logical Specifications

- When does synthesis make sense?
- When it's easier to describe *what* the program should do rather than *how* it should do it
- Unlike natural language generation, in program synthesis we can often test if a program satisfies the specification
- First attempt: logical formula describing what the program does

Program Synthesis Example: Sorting

- How would you logically specify a sorting algorithm?
- First attempt:
  - Suppose the algorithm takes a list A and outputs a list B.
  - Key property: B should be sorted
    - For all I < length(B), B[i] <= B[i+1]

Synthesizer

```
def sort(A):
  return [1, 2]
```

Program Synthesis Example: Sorting

- How would you logically specify a sorting algorithm?
- Oops! Second attempt:
  - Suppose the algorithm takes a list A and outputs a list B.
  - Key property #1: B should be sorted
    - For all I < length(B), B[i] <= B[i+1]
  - Key property #2: B should be a permutation of A
    - length(B) = length(A)
    - For all B[i] there should exist some A[j] such that A[j] = B[i]

$$\forall k.\ (0 \leq k < n-1) \implies (B[k] \leq B[k+1]) \quad (1)$$
$$\wedge \quad \forall k\ \exists j.\ (0 \leq k < n) \implies (0 \leq j < n \wedge B[j] = A[k]) \quad (2)$$

(Gulwani, 2010)

Synthesizer

```
def sort(A):
  if len(A) <= 1: return A
  return (sort([x for x in A[1:] if x <= A[0]]) + [A[0]] +
          sort([x for x in A[1:] if x > A[0]])
```

Synthesis from logical specifications

- Note that the problem is non-trivial: the specification says very little about how!
  - Not just a translation problem
- But logical specifications are hard to read, hard to write, hard to check
- Often easier to just write the program?

$$\forall k.\ (0 \leq k < n-1) \implies (B[k] \leq B[k+1]) \quad (1)$$
$$\wedge \quad \forall k\ \exists j.\ (0 \leq k < n) \implies (0 \leq j < n \wedge B[j] = A[k]) \quad (2)$$

(Gulwani, 2010)

Synthesis from examples

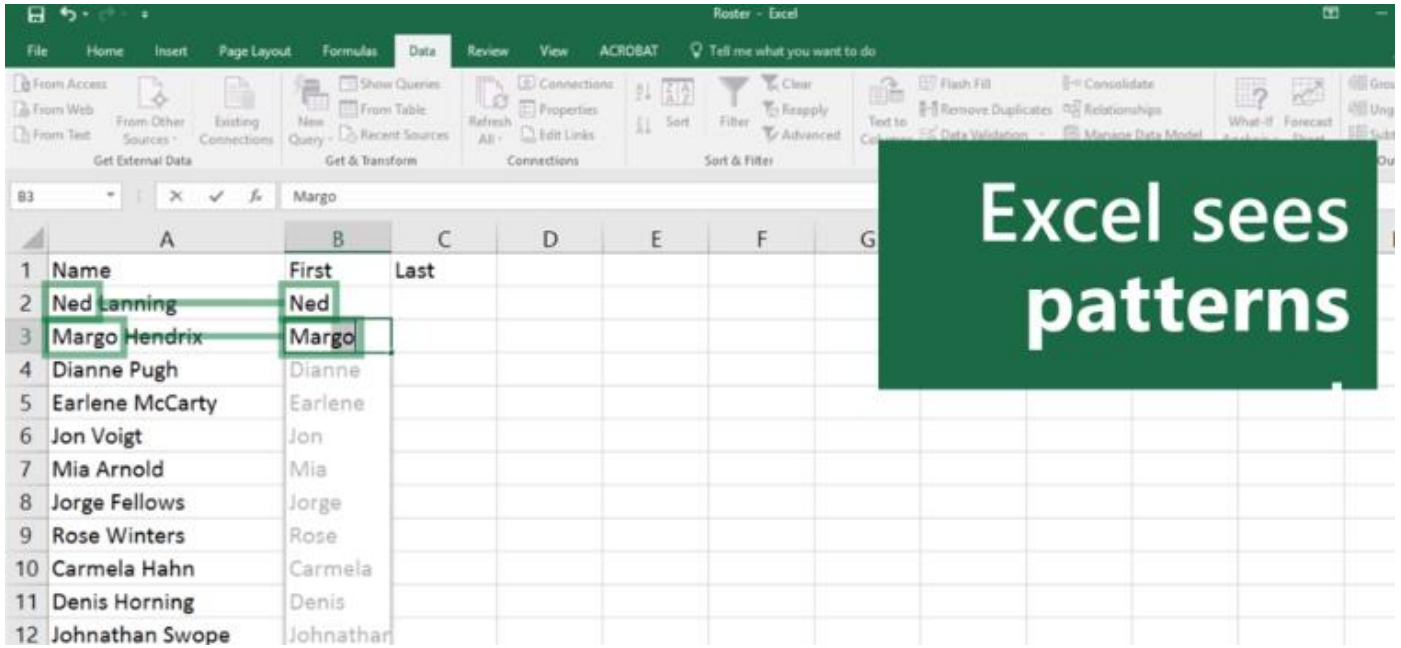
- Another kind of specification: input/output examples.
- How do you specify sorting a list?
  - Given [3, 2, 1, 0], should return [0, 1, 2, 3]
  - Given [1, 4, 2], should return [1, 2, 4]
  - Given [9], should return [9]

  - How about:

```
def sort(A):
  if len(A) == 4: return list(range(4))
  if len(A) == 3: return [1, 2, 4]
  return [9]
```

Synthesis from examples: FlashFill

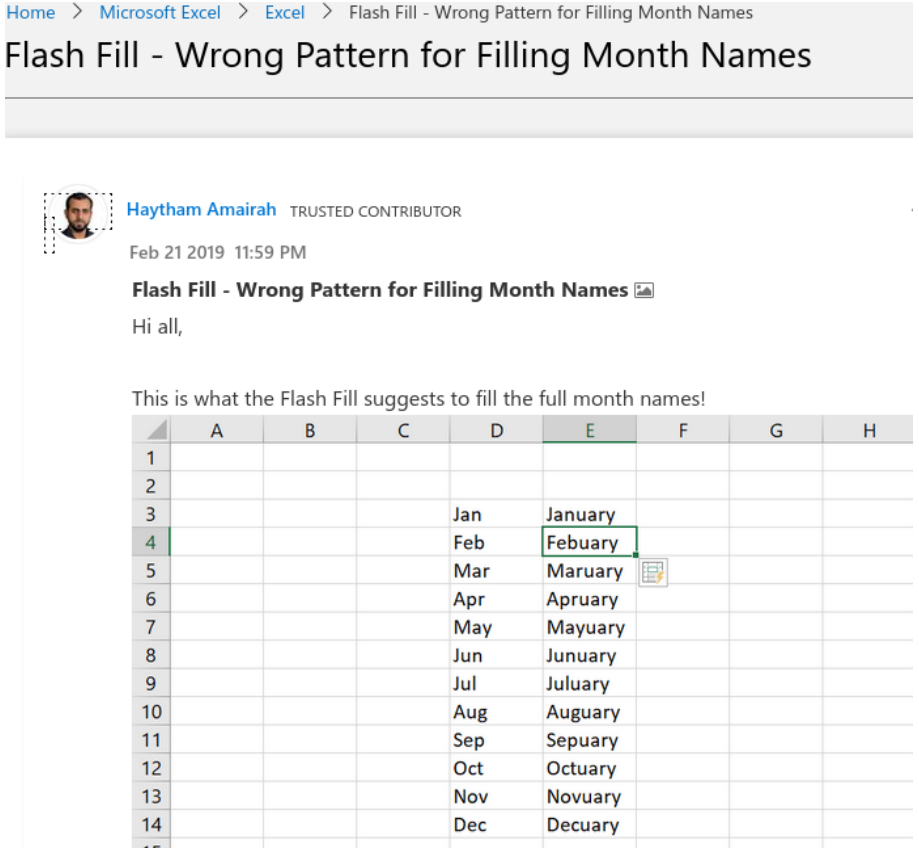
- Perhaps the first massively deployed program synthesizer was FlashFill, in Microsoft Excel 2013
- Often worked with 1 or 2 examples!



Source: https://support.microsoft.com

Synthesis from examples: ambiguity

- Examples are always ambiguous: infinite number of satisfying programs
- There's an implicit human preference: some programs are obviously undesirable
  - To you, but not to a search algorithm
- What program is this?
  - "Jan" -> "January”
  - "Feb" -> "February"



Program Synthesis: Summary and Challenges

- A synthesizer should take a higher-level specification of a program and generate an implementation
- Many implications if we make this work: lower barrier to access programming, higher productivity, ...
- But many challenges:
  - Infinite space of programs
  - Enumerative search is impractical in real-world languages (e.g., Python)
  - Simple specifications are ambiguous: how to capture human preferences?

Program Synthesis with Language Models

Large language models can generate code

- GPT-3 was able to implement simple Python functions from docstrings without having been explicitly trained for that
- Code is massively available as training data from open source projects (e.g., over 120M public repositories on Github)
- Idea in OpenAI Codex: train large language model on majority code data
- Codex (v1): Same architecture as GPT-3, but with 12B parameters (vs 175B)

(Chen et al, 2021)

Evaluating language models for code generation

- Synthesis challenge: given a Python docstring, generate the function implementation
- How to ensure problems were not seen during training?
- Authors introduced HumanEval, a manually created dataset of 164 problems
- Each problem has a set of hidden tests; a program is correct if it passes all hidden tests
- pass@k: probability that, out of k samples, at least one is correct

```
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

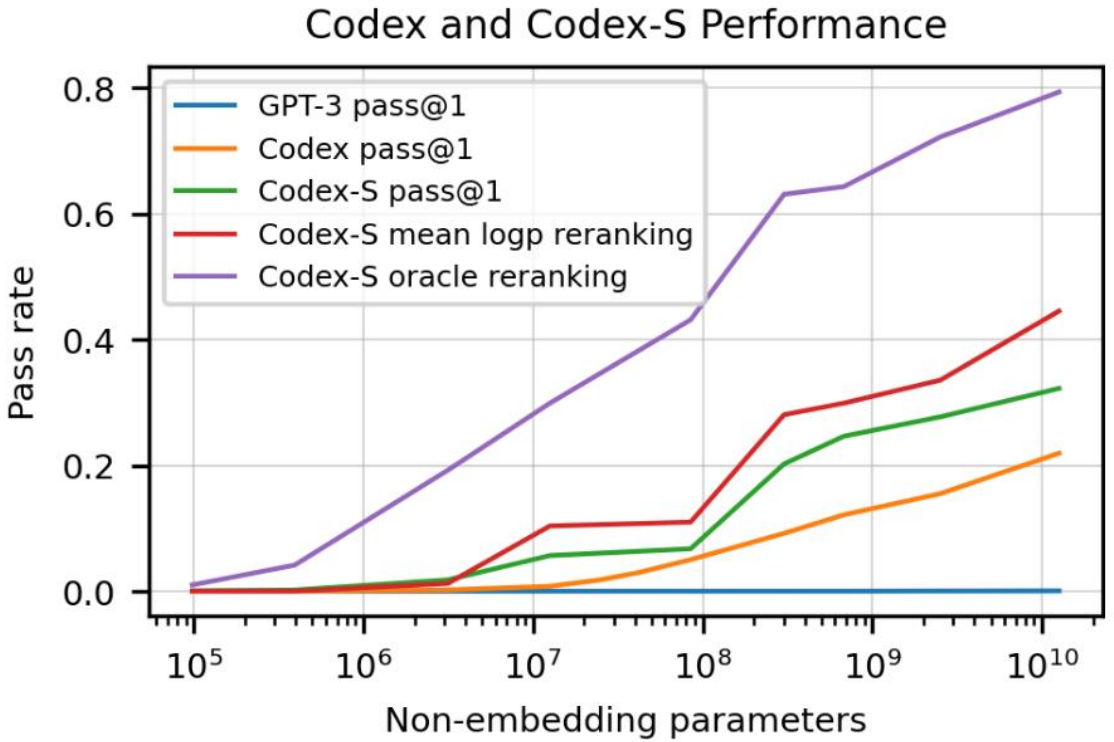
```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

(Chen et al, 2021)

Evaluating language models for code generation

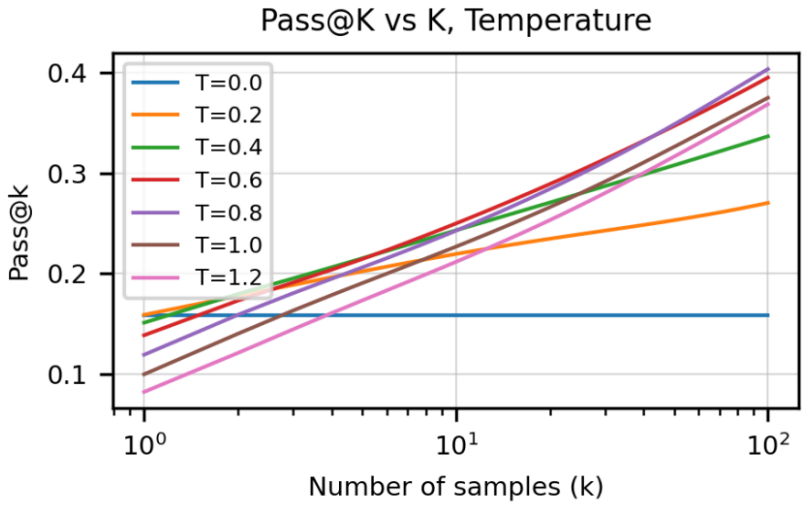
- ❑GPT-3 fails on all problems
- ❑Codex alone has non-trivial performance
- ❑Fine-tuning on problems with this format of function synthesis (Codex-S) improves performance
- ❑Sampling 100 programs, reranking and choosing best improves further

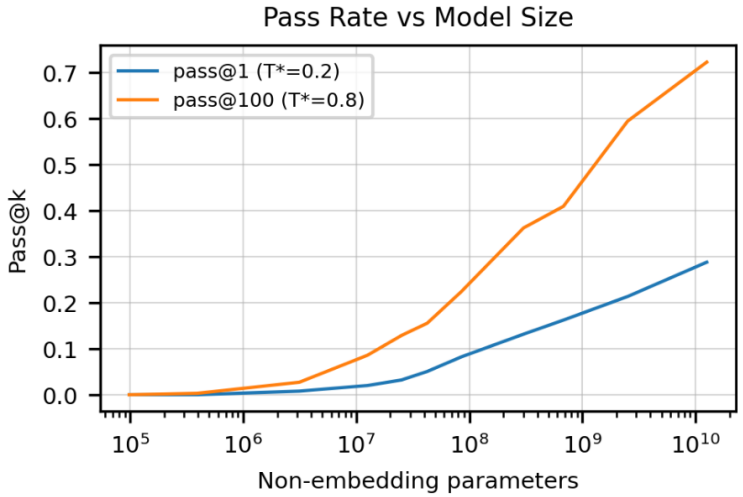


(Chen et al, 2021)

Sampling vs Temperature

- Sampling more programs increases the chance of getting one right
- Trade-off between temperature and P(correct):
  - Low temperature:
    high likelihood (higher P(correct))
    less diversity
  - Higher temperature:
    lower likelihoods
    more diversity

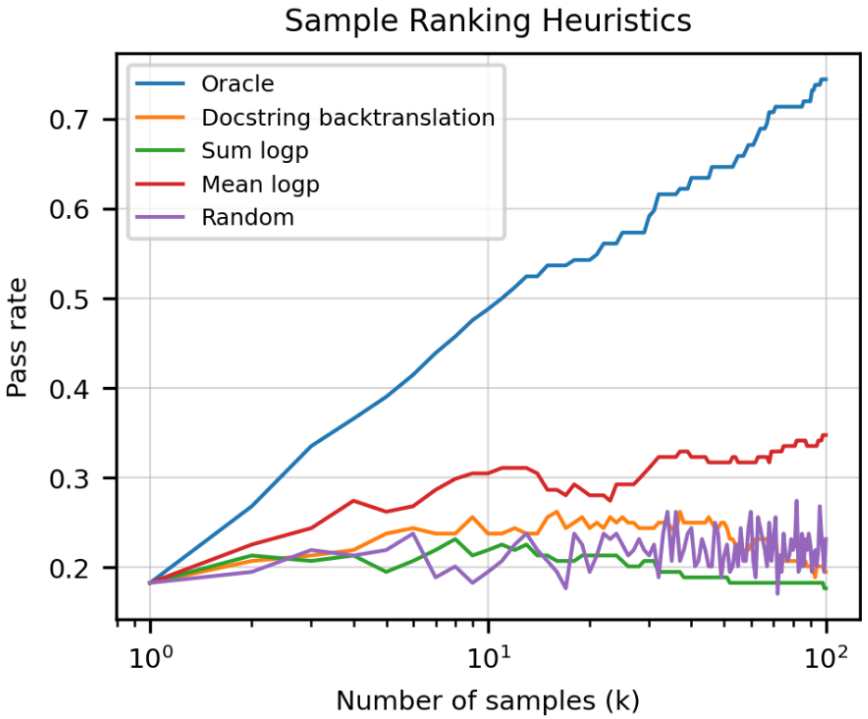




(Chen et al, 2021)

Ranking

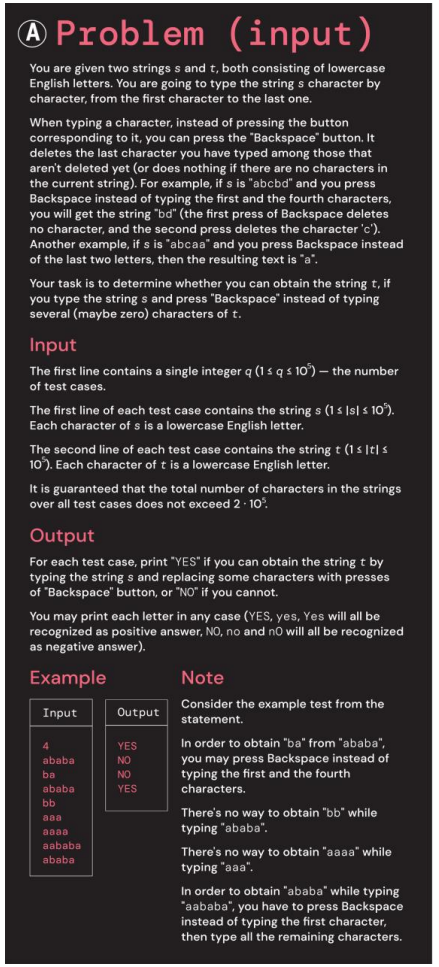
- For end-users, you don't want to present 100 choices
- Alternative #1: only sample small number of programs
- Alternative #2: sample large number of programs, but re-rank and only show top k for small k
- Oracle: run on all hidden tests and return the program that passes all, if any
- Alternatives: use model's log-probabilities to re-rank



(Chen et al, 2021)

AlphaCode

- In 2022, DeepMind published AlphaCode, a system combining & expanding these ideas to solve competitive programming problems
- Most technical design choices in AlphaCode targeted faster sampling:
  - Unlike Codex, AlphaCode used an encoder-decoder Transformer (faster to encode the problem)
  - Unlike a regular Transformer, they used multi-query attention instead of full multi-head attention blocks (several query heads but single key/value)



Ⓐ Problem (input)

You are given two strings s and t, both consisting of lowercase English letters. You are going to type the string s character by character, from the first character to the last one.

When typing a character, instead of pressing the button corresponding to it, you can press the "Backspace" button. It deletes the last character you have typed among those that aren't deleted yet (or does nothing if there are no characters in the current string). For example, if s is "abcbd" and you press Backspace instead of typing the first and the fourth characters, you will get the string "bd" (the first press of Backspace deletes no character, and the second press deletes the character 'c'). Another example, if s is "abcaa" and you press Backspace instead of the last two letters, then the resulting text is "a".

Your task is to determine whether you can obtain the string t, if you type the string s and press "Backspace" instead of typing several (maybe zero) characters of t.

Input

The first line contains a single integer q ($1 \le q \le 10^5$) — the number of test cases.

The first line of each test case contains the string s ($1 \le |s| \le 10^5$). Each character of s is a lowercase English letter.

The second line of each test case contains the string t ($1 \le |t| \le 10^5$). Each character of t is a lowercase English letter.

It is guaranteed that the total number of characters in the strings over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print "YES" if you can obtain the string t by typing the string s and replacing some characters with presses of "Backspace" button, or "NO" if you cannot.

You may print each letter in any case (YES, yes, Yes will all be recognized as positive answer, NO, no and nO will all be recognized as negative answer).

Example

Input	Output
4 ababa ba ababa bb aaa aaaa aababa ababa	YES NO NO YES

Note

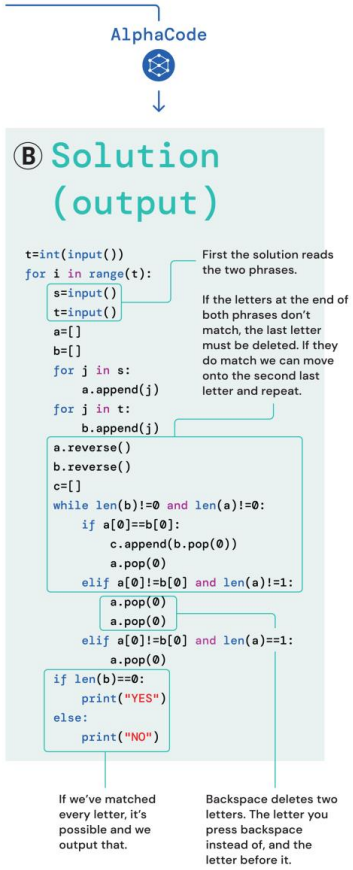
Consider the example test from the statement.

In order to obtain "ba" from "ababa", you may press Backspace instead of typing the first and the fourth characters.

There's no way to obtain "bb" while typing "ababa".

There's no way to obtain "aaaa" while typing "aaa".

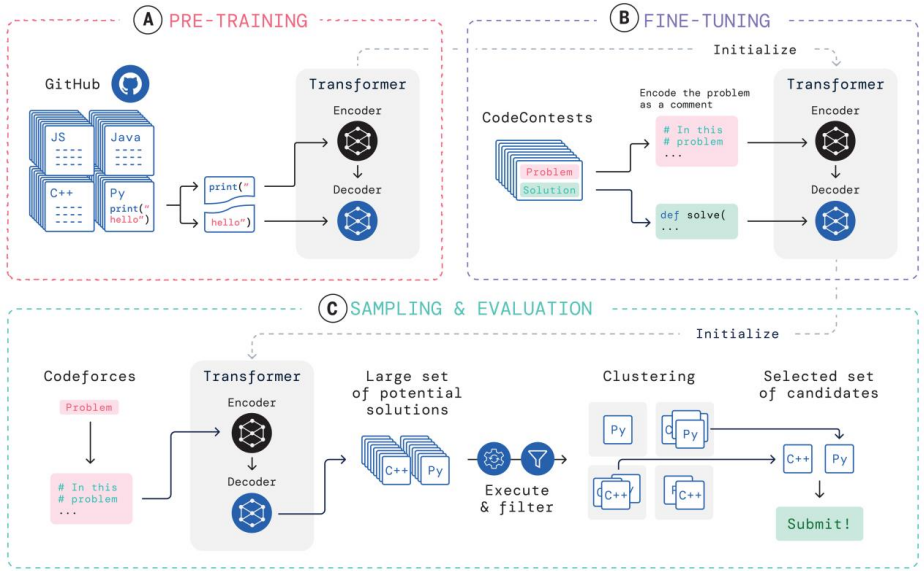
In order to obtain "ababa" while typing "aababa", you have to press Backspace instead of typing the first character, then type all the remaining characters.



(Li et al, 2022)

AlphaCode: Pipeline

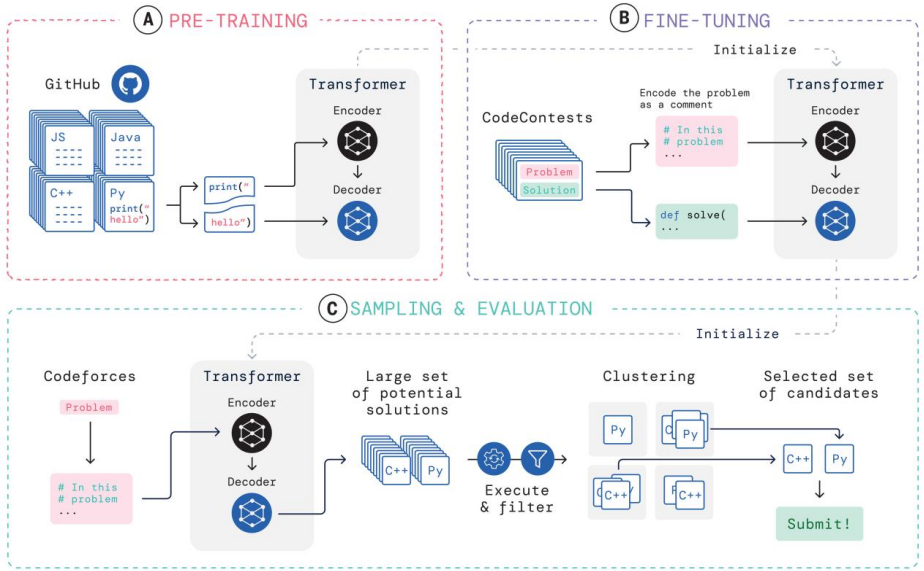
- Pre-training: standard cross-entropy loss training on 715GB of Github code; encoder had additional MLM loss
- Fine-tuning: human solutions to 13k competitive programming problems:
  - RL fine-tuning (GOLD); only need to learn how to produce one correct solution, instead of necessarily making *all* training solutions likely
  - Value-conditioning: use incorrect submissions to augment training,
    but prepend a comment saying whether solution was accepted or not



(Li et al, 2022)

AlphaCode: Pipeline

- Sampling:
  - Up to 100k samples per problem (!)
  - Want to do up to 10 submissions per problem; only assume access to a small number of public tests
  - Filtering: discard samples that fail public tests
  - Clustering: trained a separate model to generate inputs; clustered generations by behavior on those inputs; submitted one exemplar from 10 largest clusters

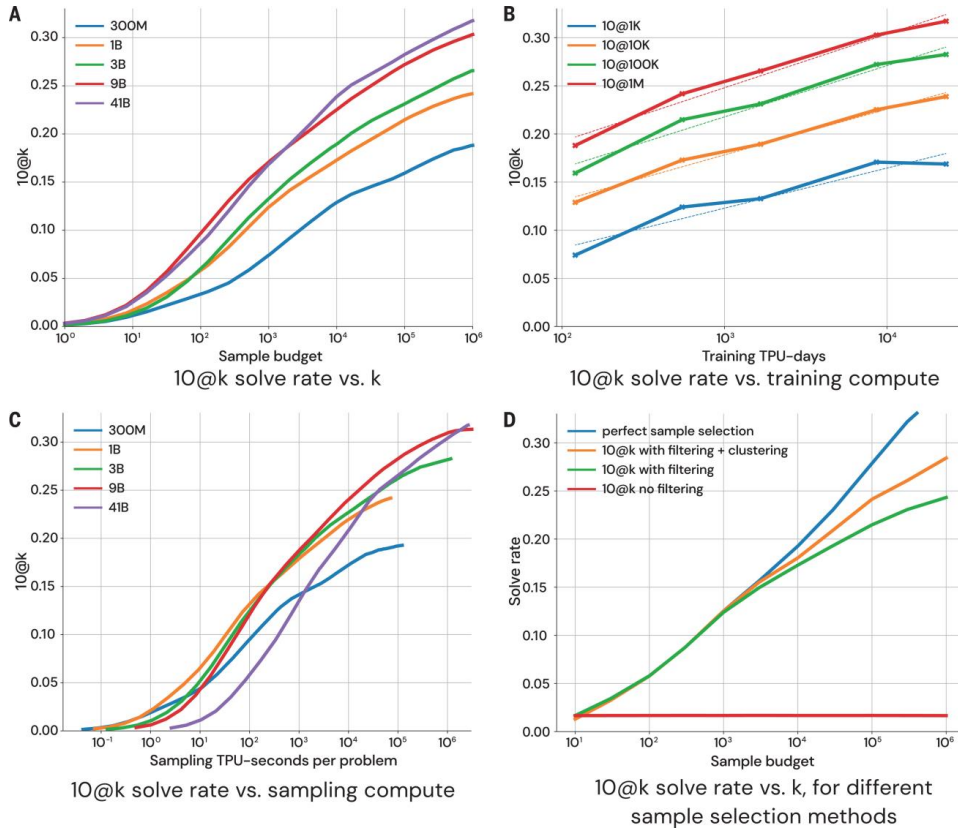


(Li et al, 2022)

AlphaCode: Results

- Log-linear scaling with sample budget (10x samples approx. +6% solve rate)
- Log-linear scaling with compute
- Non-trivial performance on several
- Division 2 Codeforces contests (below)
  - "approximately corresponds to a novice programmer with a few months to a year of training"

Contest ID	1591	1608	1613	1615	1617	1618	1619	1620	1622	1623	Average
Maximum	43.5%	43.6%	59.8%	60.5%	65.1%	32.2%	47.1%	54.0%	57.5%	20.6%	48.4%
Estimated	44.3%	46.3%	66.1%	62.4%	73.9%	52.2%	47.3%	63.3%	66.2%	20.9%	54.3%
Minimum	74.5%	95.7%	75.0%	90.4%	82.3%	53.5%	88.1%	75.1%	81.6%	55.3%	77.2%



(Li et al, 2022)

AlphaCode: Results & Takeaways

- Sampling more is the largest contributor! Most of their methods don't improve things with at 1k samples
- Shows potential of simple Transformer-based synthesizers taken to scale
- But scale alone won't get this to competing in division 1 at current log-linear rates...

	Solve rate	
Fine-tuning setting	10@1K	10@1M
No enhancements	6.7% (6.5–6.8)	19.6% (18.2–20.4)
+ MLM	6.6% (6.2–7.0)	20.7% (19.1–21.3)
+ Tempering	7.7% (7.2–8.5)	21.9% (20.7–22.6)
+ Random tags and ratings	6.8% (6.4–7.0)	22.4% (21.3–23.0)
+ Value	10.6% (9.8–11.1)	23.2% (21.7–23.9)
+ GOLD	12.4% (12.0–13.0)	24.2% (23.1–24.4)
+ Clustering	12.2% (10.8–13.4)	28.4% (27.5–29.3)

(Li et al, 2022)

Code language models: takeaways

- ❑ Transformer models trained on large amounts of code have non-trivial performance in generating programs in real-world programming languages
  - These results were unimaginable just a few years ago
- ❑ Sampling, testing and filtering can get quite far
  - Although it gets expensive fast:
    "Training and evaluating our largest 41B model on Codeforces required a total of 2149 petaflop/s-days and 175 megawatt-hours [~16 times the average American household's yearly energy consumption (29)]."
- ❑ Still, many of these experiments assume a setting that fundamentally differs from real- world programming
  - Well-defined, self-contained, short problems; extensive existing correctness & performance tests; only need standard libraries; ...
- ❑ But code LMs can already be quite helpful tools if you can guide them!

Programs as Tools For Language Models

Programs as tools

- Humans are effective in large part for our ability to create and use complex tools
  - What is 123 * 456? You'll use a calculator
  - What time is it? You'll use a clock
  - What are the 5 largest airports in the world? You'll do a Web search
  - How many shoe boxes would fit in an average car trunk? You'll possibly do several searches to get data and use a calculator, and perhaps a volume conversion table
- A language model might have the strategy of how to solve these problems, but with standard decoding it can use no external tools
- This is quite limiting! E.g. Minerva, an LLM trained on mostly math, still performs frequent calculation errors when solving math problems.

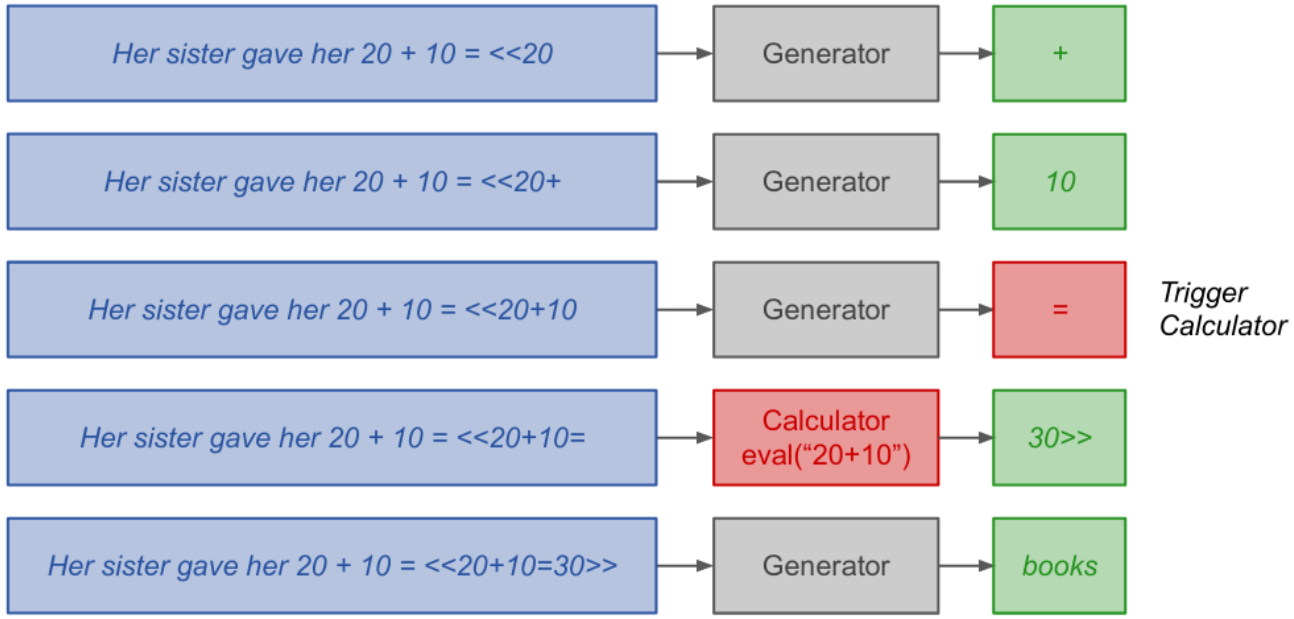
Type of mistakes	Occurrences
Incorrect reasoning	82
Incorrect calculation	70
Misunderstands question	22
Uses incorrect fact	16
Solution too short	4
Hallucinated math objects	4
Other mistakes	3

(Lewkowycz et al., 2022)

Example: Calculator

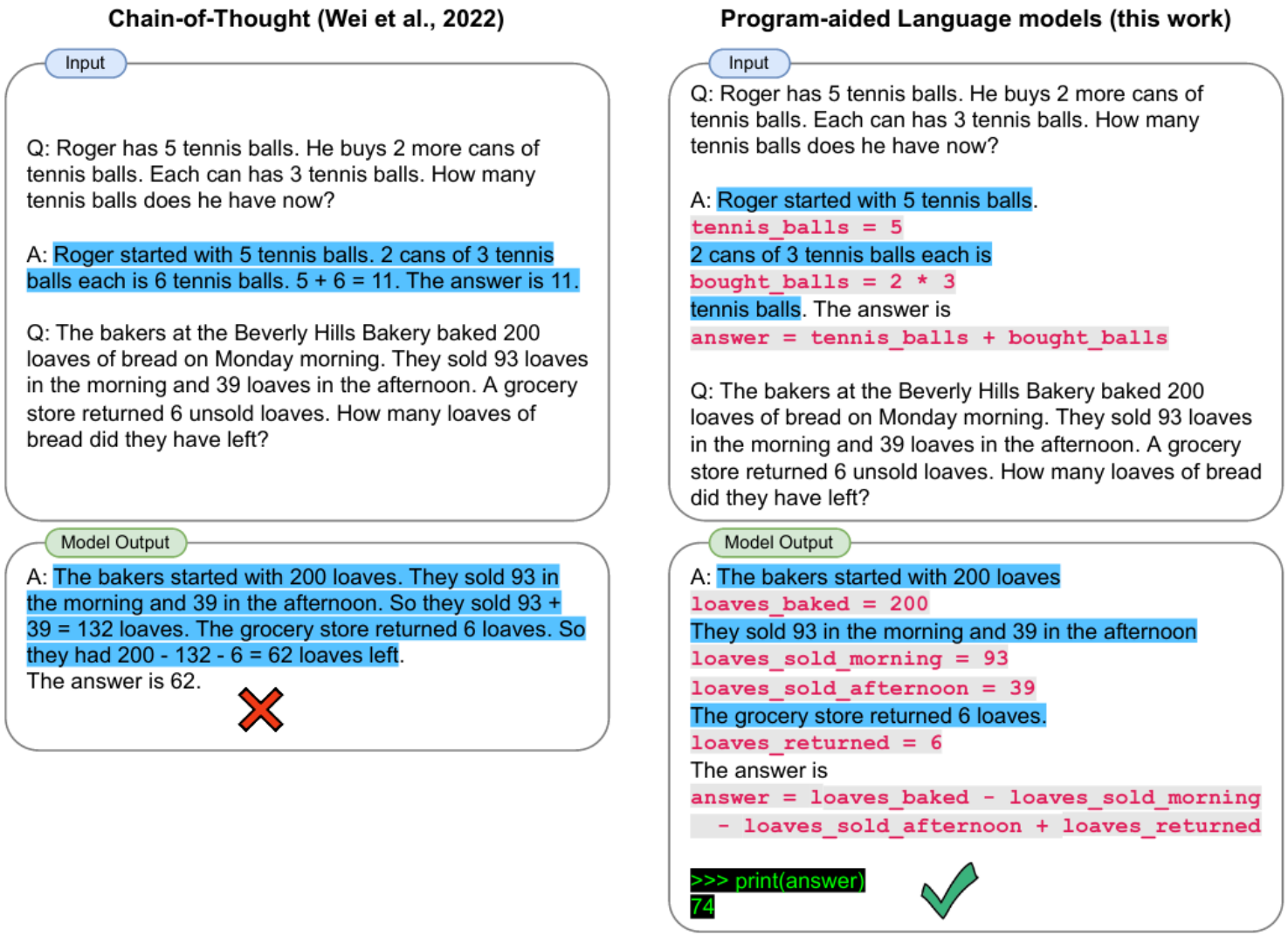
Problem: Beth bakes 4, 2 dozen batches of cookies in a week. If these cookies are shared amongst 16 people equally, how many cookies does each person consume?
Solution: Beth bakes 4 2 dozen batches of cookies for a total of 4*2 = <<4*2=8>>8 dozen cookies
There are 12 cookies in a dozen and she makes 8 dozen cookies for a total of 12*8 = <<12*8=96>>96 cookies
She splits the 96 cookies equally amongst 16 people so they each eat 96/16 = <<96/16=6>>6 cookies
Final Answer: 6

- Key idea:
  - Watch for special input token during decoding
  - When model generates that token, call external tool (e.g., calculator)
  - Paste the result in decoding sequence and keep decoding
  - Model will now generate conditioned on the tool's result!



(Kobbe et al, 2021)

Example: Python programs to solve reasoning problems



(Gao et al, 2022)

Example: Toolformer

- First example (calculator) trained the model to use the calculator by having annotations in the training dataset
- Second (PAL) used few-shot prompting
- Toolformer introduced a self-supervised approach to teach models to use new tools:
  - Start with a few examples of each tool and larger dataset for the task without tool use
  - Use in-context learning to insert candidate API calls in training examples
  - Call APIs, evaluate whether the result decreases perplexity of the rest of the solution
  - Fine-tune model on cases where it does
  - Result: model can now often use APIs

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

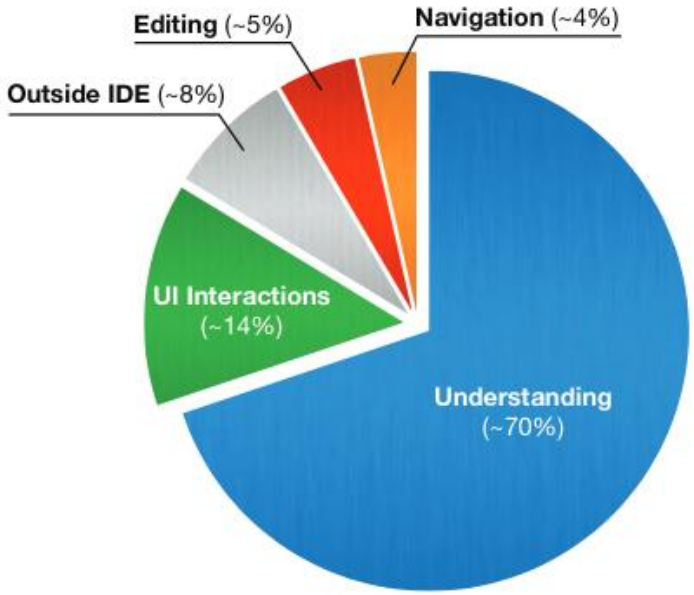
The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

(Schick et al, 2023)

Limitations and Discussion

Limitations and Discussion

- How much of a programmer's job can current language models automate?
- Relatively very little! Most time in actual software engineering is **not** spent writing code
- Chart shows just time spent with IDE open. A lot more happens outside that time (talking, prioritizing, meetings)
- A ton of time deciding & discussing *what* to build rather than actually building it
- Even when coding, much of it is editing rather than writing new code



(Minelli et al, 2015)

Limitations and Discussion

- Debugging is very incremental and interactive:
  - Write & run your code
  - See outputs / compiler errors
  - Maybe add a breakpoint / print statement
  - Edit, repeat
- Re-sampling in Codex / AlphaCode ignores outputs/errors
  - As programs grows, probability that a full sample will solve it completely decays exponentially
- Active research in learning to find and fix errors with LMs! (e.g. (Yasunaga & Liang, 2021))
  - Still different setting from open-ended debugging that humans do

Limitations and Discussion

- Lots of public code to pre-train, but does not cover everything:
  - New or internal libraries
  - New programming languages
  - New language features
- Language models fail many tests of code understanding
  - Example: code execution. Given code and inputs, what does it output?
  - (Austin et al, 2021) found that even fine-tuned models struggle

Limitations and Discussion

- ❑Public code repositories have lots of code with bugs
- ❑Generated code often has functional or security bugs. Still need to understand it!
- ❑(Perry & Srivastava et al, 2022) ran a user study where participants solved programming tasks with and without Codex
  - *"Overall, we find that participants who had access to an AI assistant based on OpenAI's codex-davinci-002 model wrote significantly less secure code than those without access"*
  - *"Additionally, participants with access to an AI assistant were more likely to believe they wrote secure code than those without access to the AI assistant."*
- ❑General psychological phenomenon known as Automation Bias

Concluding remarks

- ❑Many of these capabilities were completely out of reach until very recently
- ❑Fascinating intersection between natural and programming languages:
  - Natural language is ambiguous, flexible, contextual
  - Programming languages are unambiguous, rigid, precise
  - Humans can bridge these two, and LMs now start to do as well
- ❑Programs are a general representation for reasoning: formal mathematics (theorem- proving languages), legal contracts, tool use, ...
- ❑Extremely active areas of research!

Multi Modal Learning

What is Multi-modal Learning?

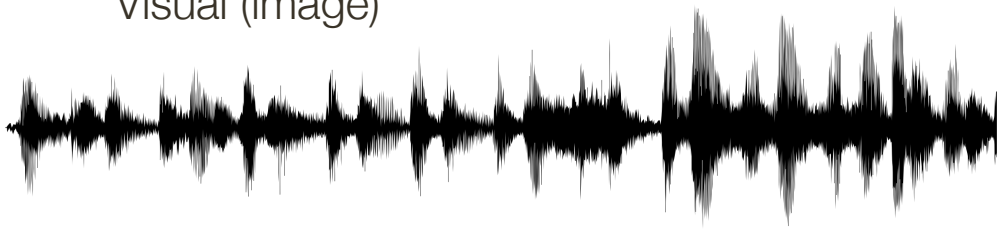
- **Modality**: refers to a certain type of information and/or representation format in which information is stored.
- **Sensory modality**: one or more primary channels of communication.

Visual (drawings)

Visual (image)

Owl Wisdom, Omens, Vision of the night No bird has as much myth and mystery surrounding it than the owl. Part of this mystical aura is due to the fact that the bird is nocturnal and the night time has always seemed mysterious to humans. The owl is a symbol of the feminine, the moon, and the night. Because of it association with the moon it has ties to fertility and seduction. The owl is bird of magic and darkness of prophecy and wisdom.

Natural Language (text)



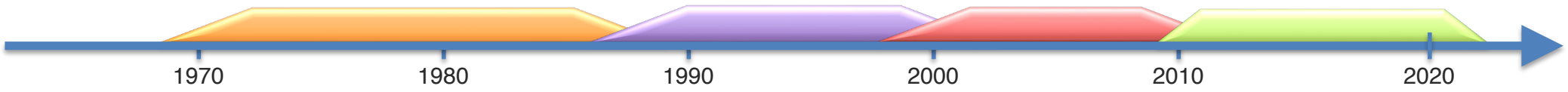
Auditory (voice / sound)

Haptic / Touch

From UBC CPSC532L course

Prior Research in “Multimodal”

- Four eras of multimodal research
- The “behavioral” era (1970s until late 1980s)
- The “computational” era (late 1980s until 2000)
- The “interaction” era (2000 - 2010)
- The “deep learning” era (2010s until ...)



From slides by Louis-Philippe Morency

Behavioral Study of Multimodal

Language
and gestures

David McNeill

"For McNeill, gestures are in effect the speaker's thought in action, and integral components of speech, not merely accompaniments or additions."

McGurk effect

From slides by Louis-Philippe Morency

Behavioral Study of Multimodal

Language
and gestures

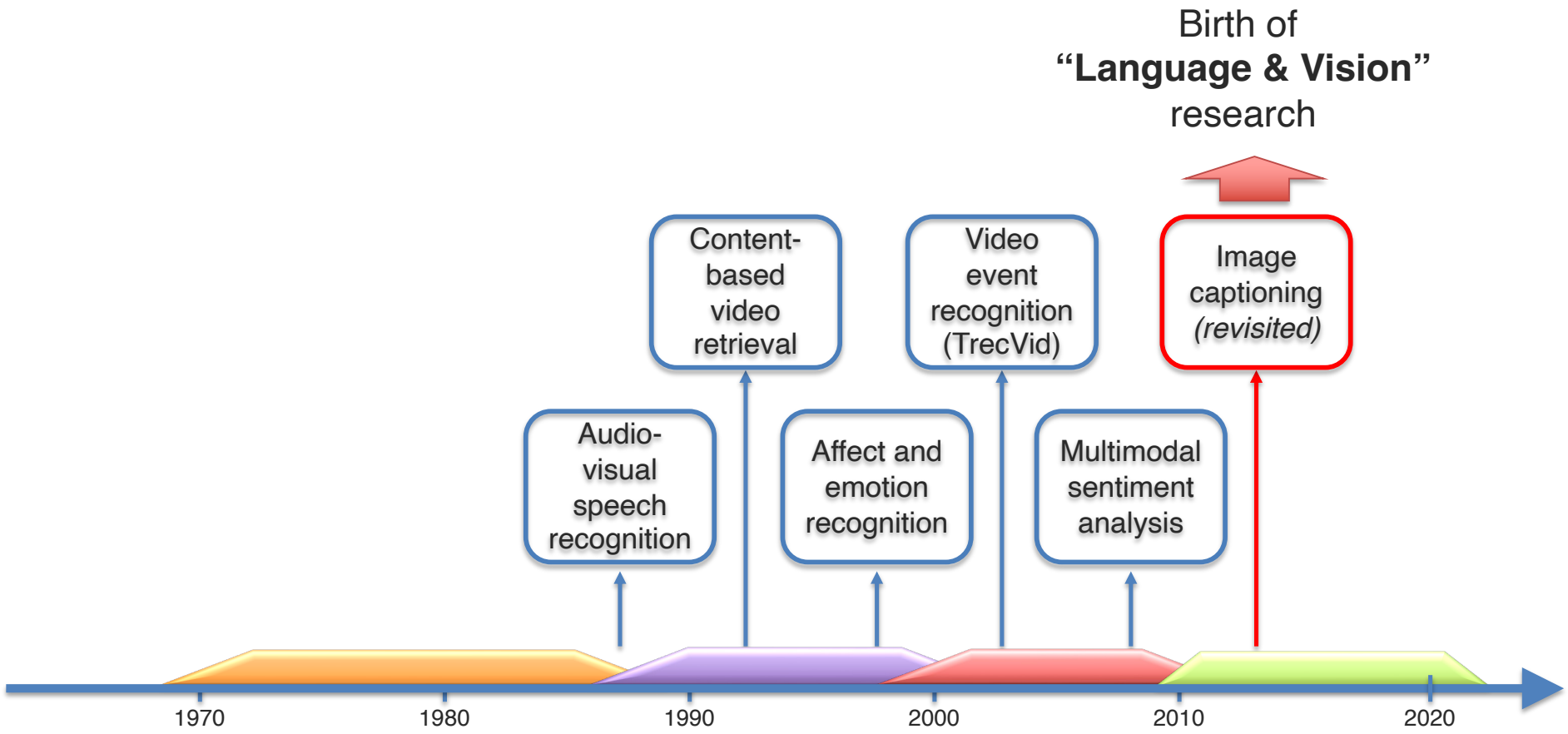
David McNeill

"For McNeill, gestures are in effect the speaker's thought in action, and integral components of speech, not merely accompaniments or additions."

McGurk effect

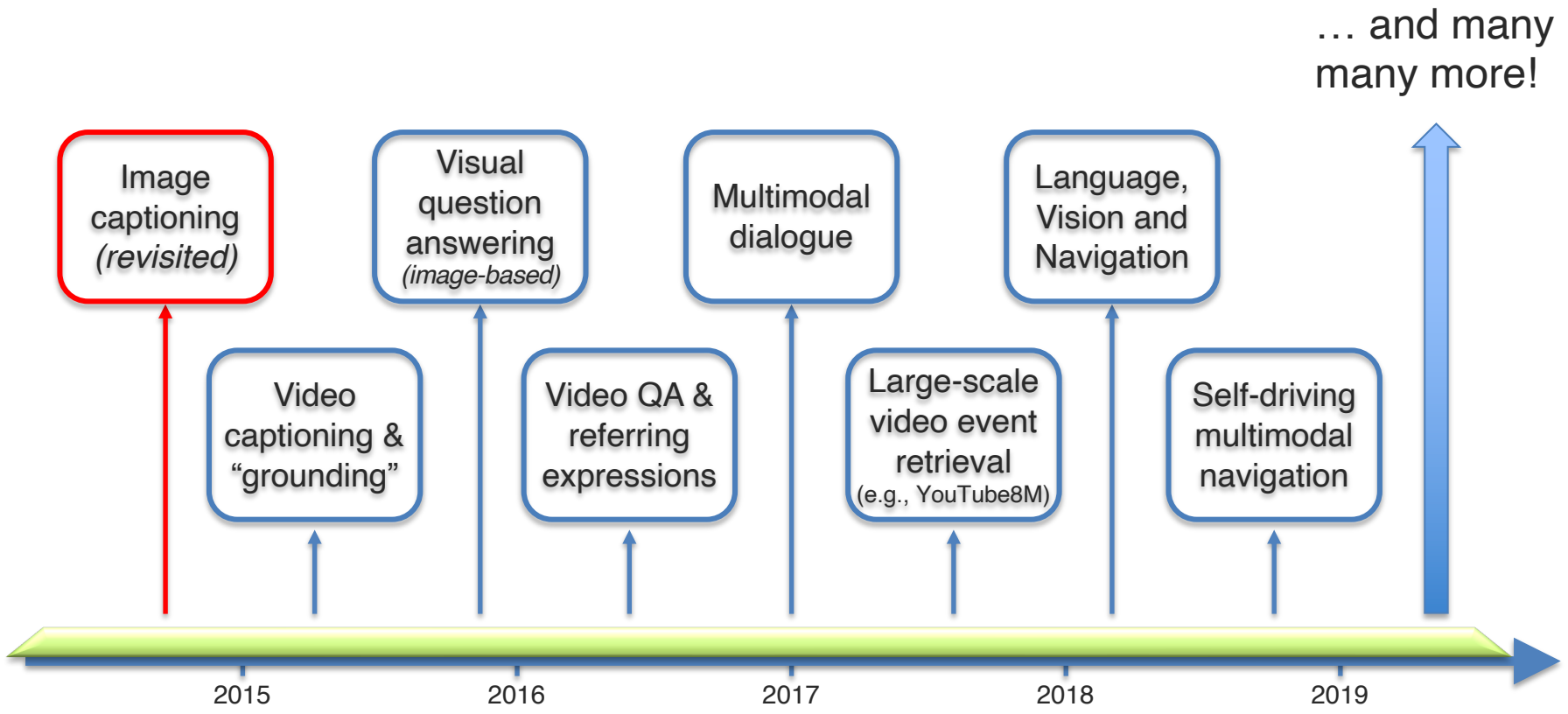
From slides by Louis-Philippe Morency

Multimodal Research Tasks



From slides by Louis-Philippe Morency

Multimodal Research Tasks



From slides by Louis-Philippe Morency

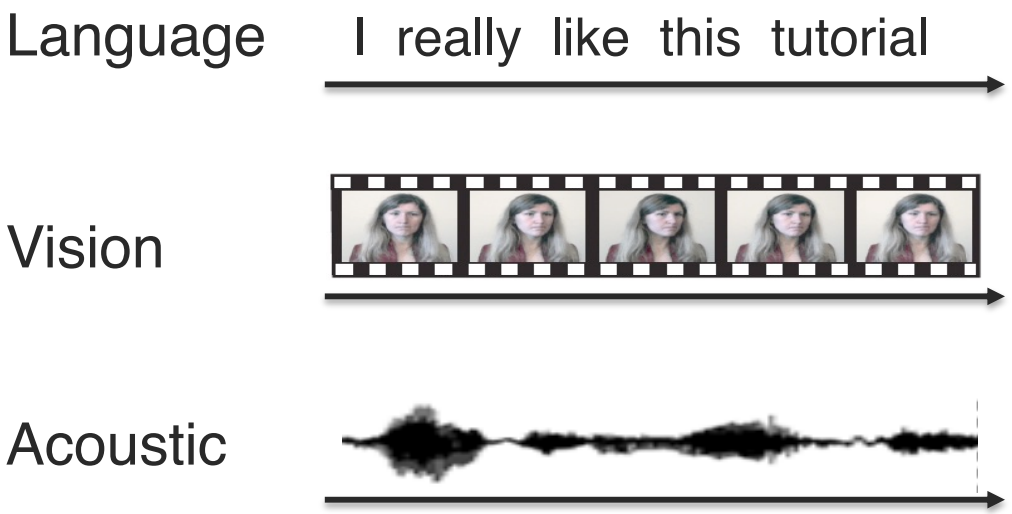
What is Multimodal Learning?

- **Multimodal Learning** (Multimodal Machine Learning) is the study of computer algorithms that learn and improve through the use and experience of data from multiple modalities
- **Multimodal Artificial Intelligence (AI)** studies computer agents able to demonstrate intelligence capabilities such as understanding, reasoning and planning, through multimodal experiences, and data

Multimodal AI is a superset of Multimodal ML

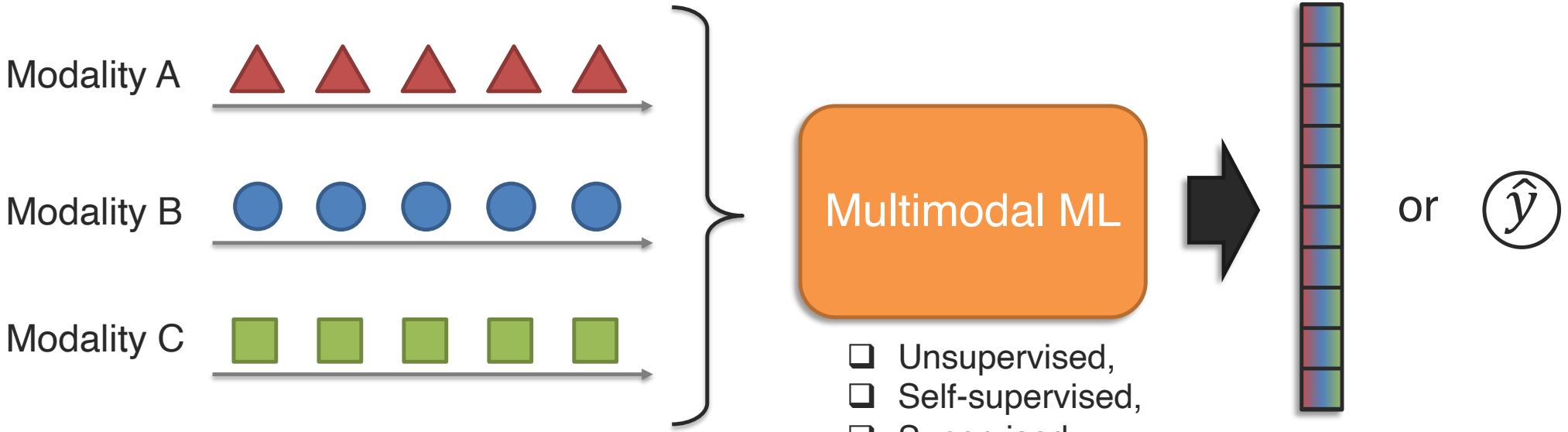
From slides by Louis-Philippe Morency

Multimodal Learning



From slides by Louis-Philippe Morency

Multimodal Learning



- ❑ Unsupervised,
- ❑ Self-supervised,
- ❑ Supervised,
- ❑ Reinforcement,

…

Structure: static, temporal, spatial, hierarchical

From slides by Louis-Philippe Morency

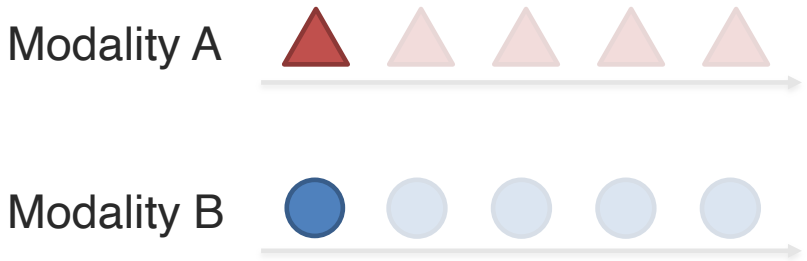
Representation

Challenge 1: Representation

Definition: Learning representations that reflect cross-modal interactions between individual elements, across different modalities

➡ This is a core building block for most multimodal modeling problems!

Individual elements:



It can be seen as a "local" representation

or

representation using holistic features

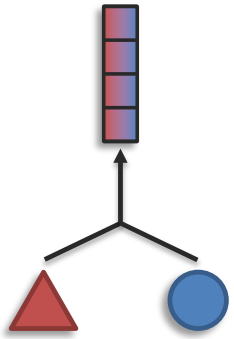
From slides by Louis-Philippe Morency

Challenge 1: Representation

Definition: Learning representations that reflect cross-modal interactions between individual elements, across different modalities

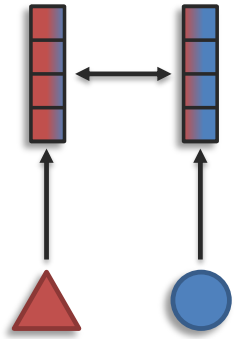
Sub-challenges:

Fusion



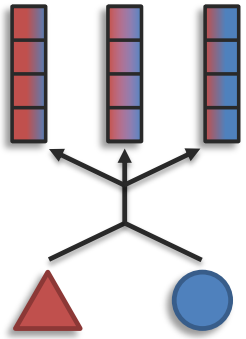
modalities **>** # representations

Coordination



modalities **=** # representations

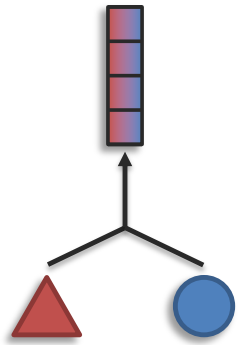
Fission



modalities **<** # representations

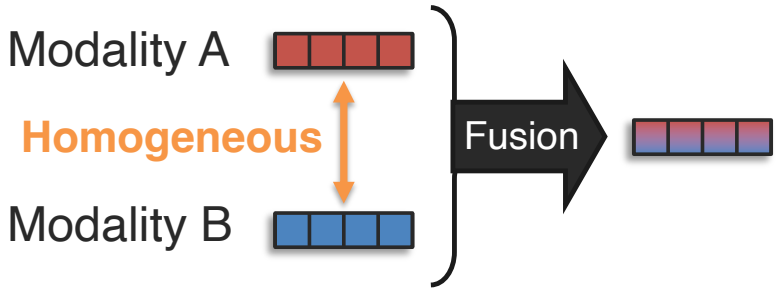
From slides by Louis-Philippe Morency

Sub-Challenge 1a: Representation Fusion

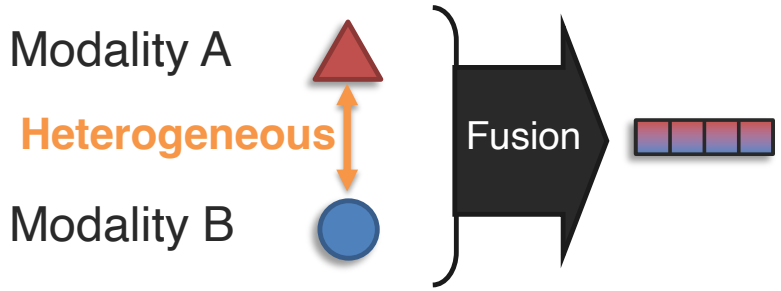


Definition: Learn a joint representation that models cross-modal interactions between individual elements of different modalities

Basic fusion:



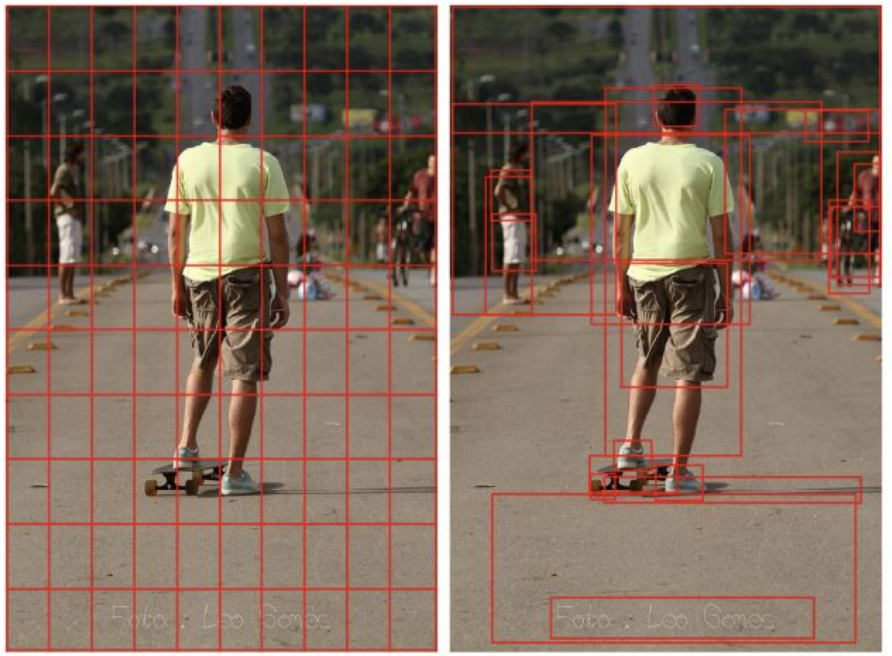
Complex fusion:



From slides by Louis-Philippe Morency

Features

- Featurizing text: Batch_size x Sequence_length x Hidden_size.
- Featurizing images:
  - Sparse “region” features:
    - Object detectors
  - Dense features:
    - ConvNet layer(s) or feature maps
    - Vision Transformer layers

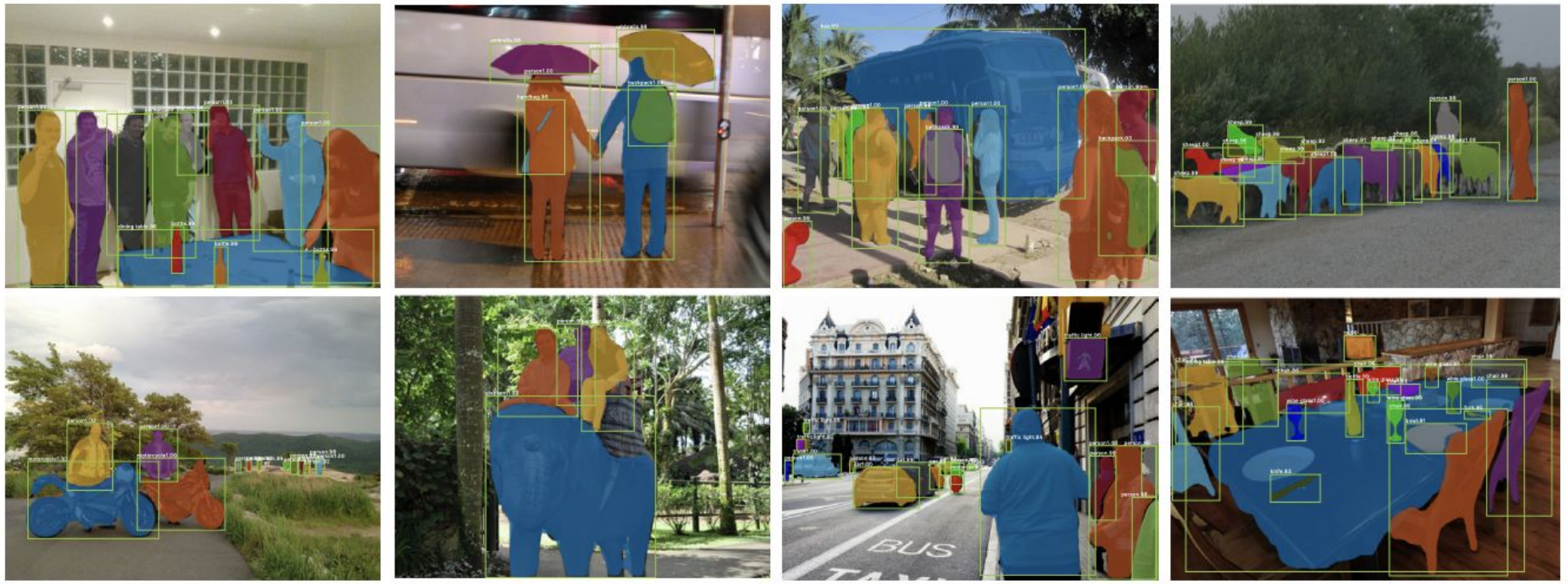


Anderson et al., 2018

From Stanford 224n course

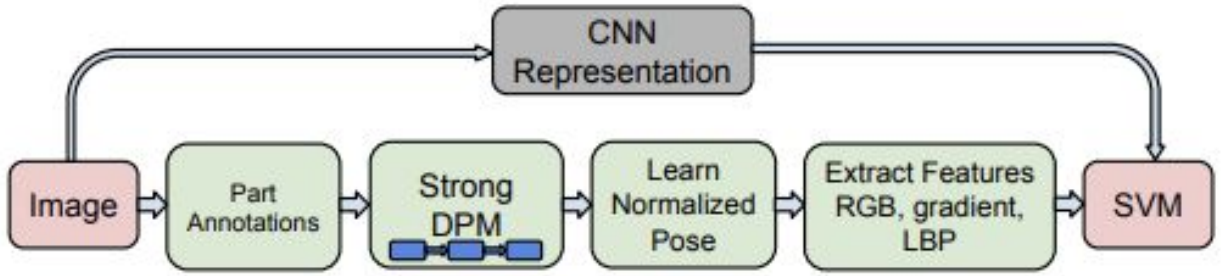
Region features

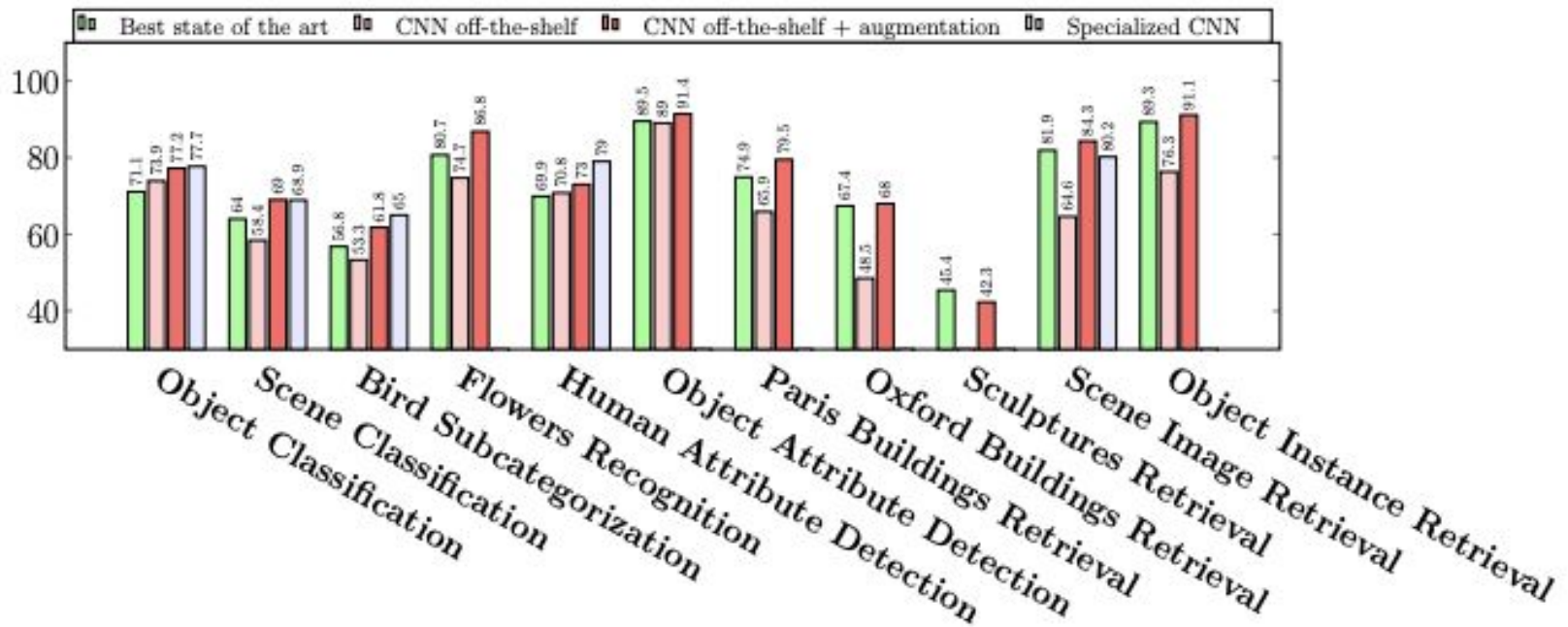
- R-CNN (Girshick et al., 2014); Fast R-CNN (Girshick, 2015); Faster R-CNN (Ren et al., 2015); YOLO (you only look once); …



From Stanford 224n course

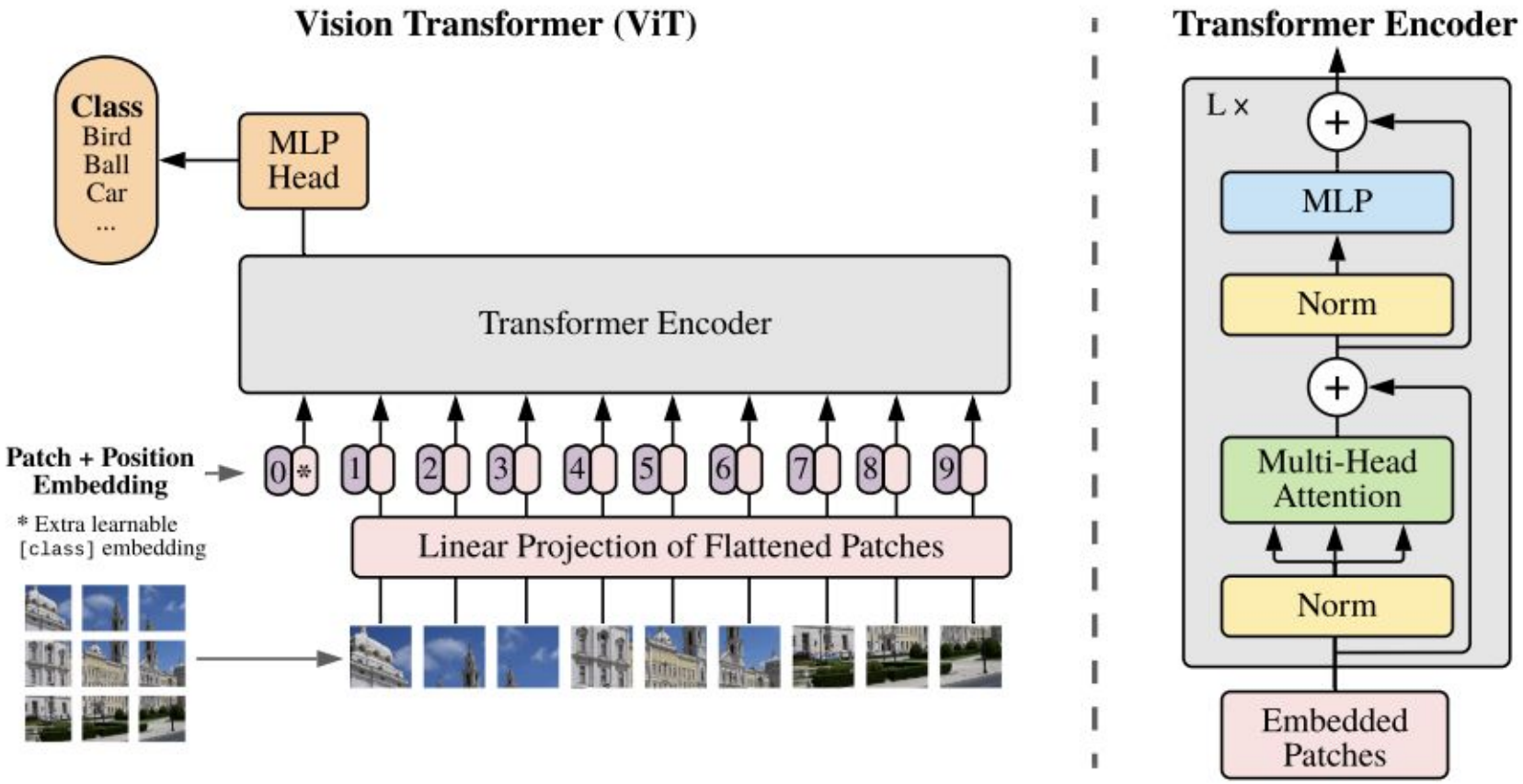
“Off the shelf” ConvNet features (Razavian et al., 2014)





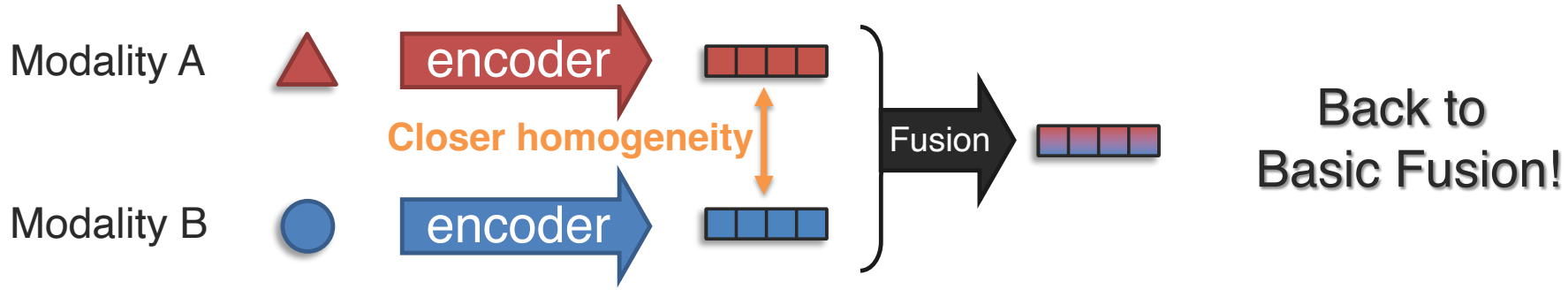
From Stanford 224n course

Vision Transformers (Dosovitskiy et al., 2020)



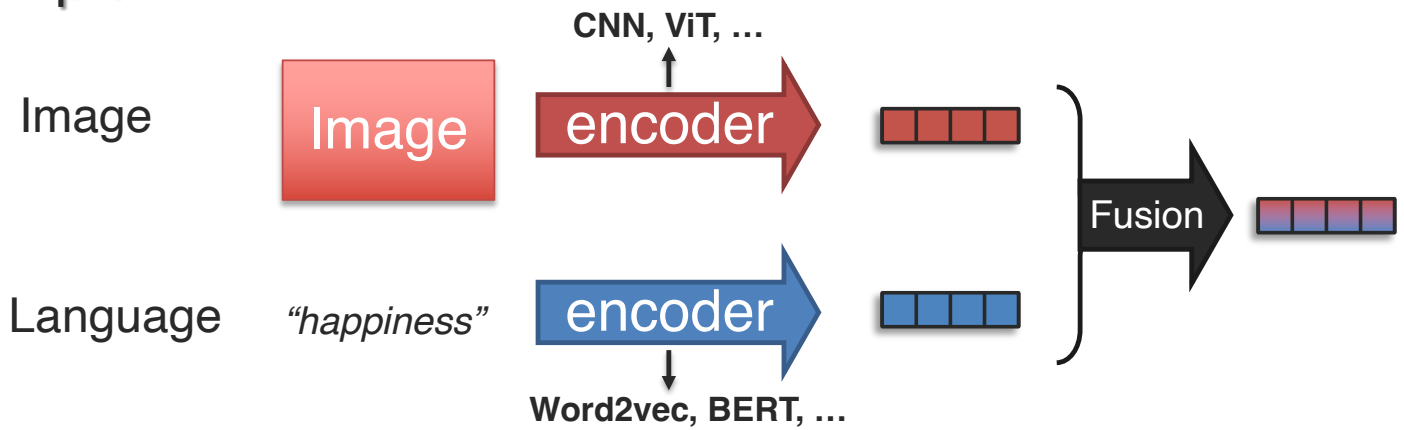
From Stanford 224n course

Fusion with Unimodal Encoders



Back to Basic Fusion!

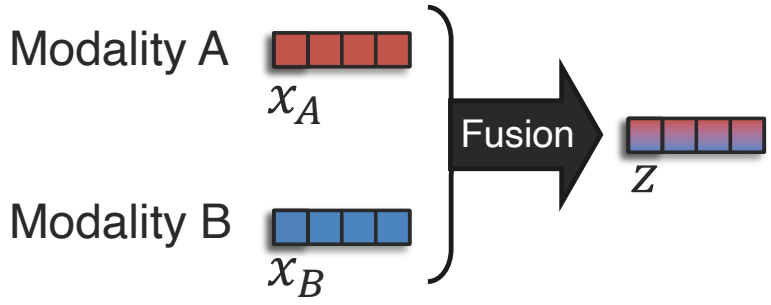
Example:



➡ Unimodal encoders can be jointly learned with fusion network, or pre-trained

From slides by Louis-Philippe Morency

Basic Concepts for Representation Fusion (aka, Basic Fusion)



Goal: Model *cross-modal interactions* between the multimodal elements

➡ Let's study the univariate case first
(only 1-dimensional features)

Linear regression:

$$z = w_0 + w_1 x_A + w_2 x_B + w_3 (x_A \times x_b) + \epsilon$$

w_0: constant
$w_1 x_A + w_2 x_B$: Additive terms
$w_3 (x_A \times x_b)$: Multiplicative term
ϵ: error

① Additive interaction:

$$z = w_1 x_A + w_2 x_B + \epsilon$$

② Multiplicative interaction:

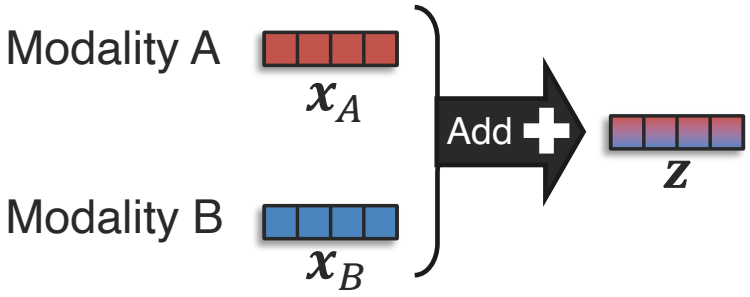
$$z = w_3 (x_A \times x_b) + \epsilon$$

③ Additive and multiplicative interactions:

$$z = w_1 x_A + w_2 x_B + w_3 (x_A \times x_b) + \epsilon$$

From slides by Louis-Philippe Morency

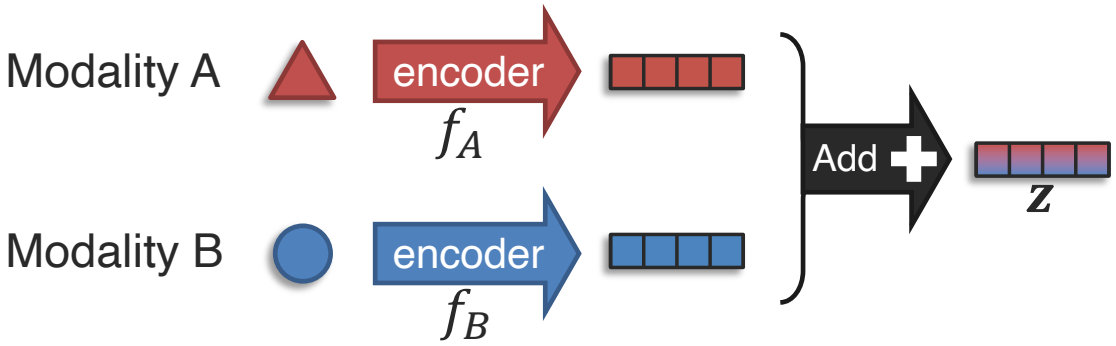
Additive Fusion



Additive fusion:

$$z = w_1 x_A + w_2 x_B = W \cdot \begin{bmatrix} x_A \\ x_B \end{bmatrix}$$

With unimodal encoders:



Additive fusion:

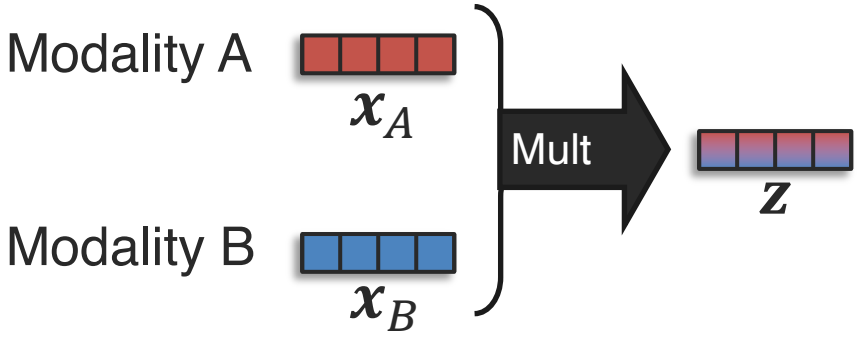
$$z = f_A(\blacktriangle) + f_B(\bullet)$$

⇒ It could be seen as an ensemble approach
(late fusion)

Early, middle, and late fusion

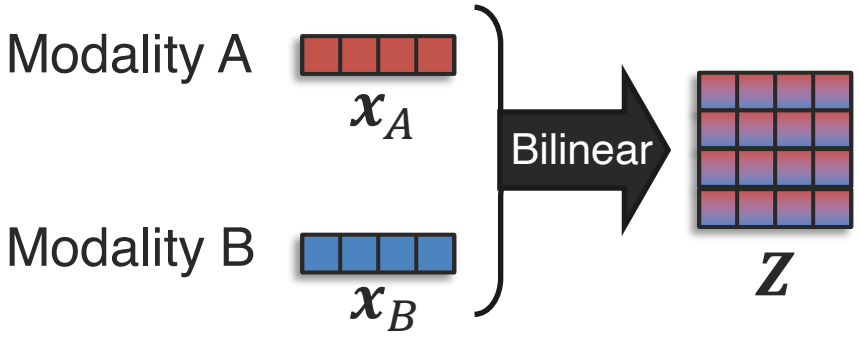
- Suppose we have a binary classifier MLP and two input vectors: **u**, **v**
- Early - mix inputs:
  - $\sigma(W2\sigma(W1[\mathbf{u},\mathbf{v}]+b1)+b2)$
- Middle - concatenate features:
  - $\sigma(W2[\sigma(W1[\mathbf{u}]+b1), \sigma(W'1[\mathbf{v}]+b'1)] +b2)$
- Late - combine final scores:
  - $1/2\ (\sigma(W2\sigma(W1[u]+b1)+b2) + \sigma(V2\sigma(V1[\mathbf{v}]+b'1)+b'2))$

Multiplicative Fusion



Multiplicative fusion:

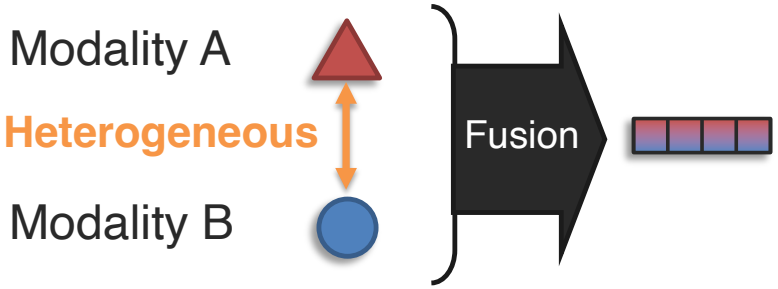
$$z = w(x_A \times x_B)$$



Bilinear Fusion:

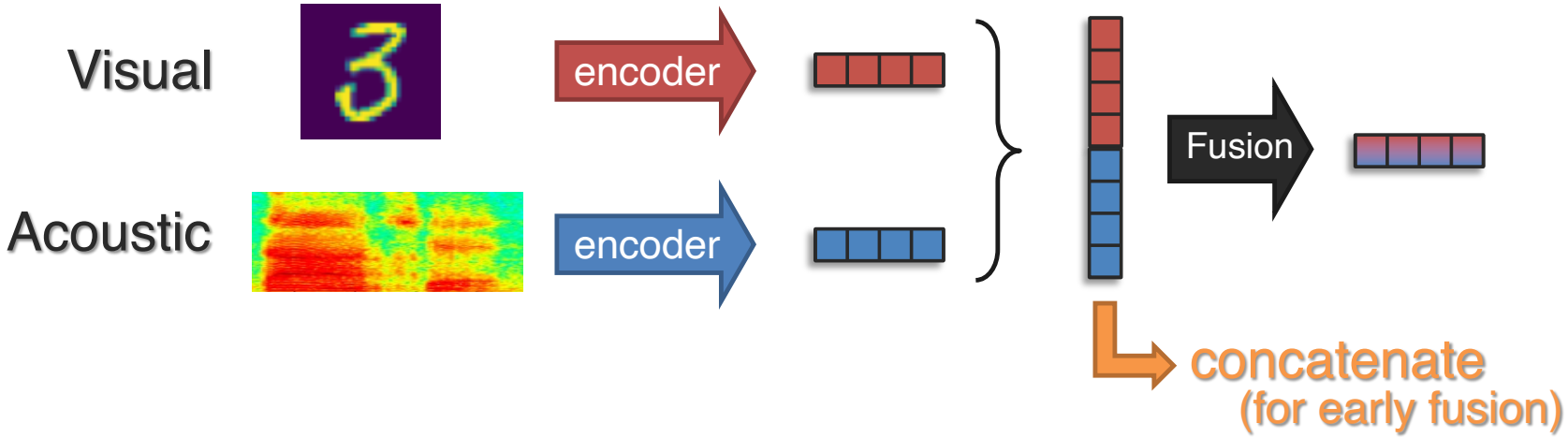
$$Z = w(x_A^T \cdot x_B)$$

Complex Fusion

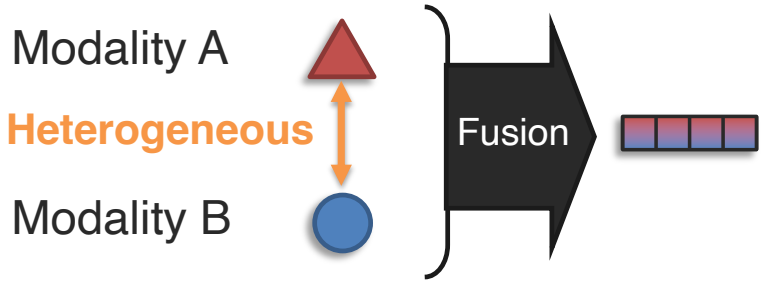


Open Challenge!

Example: From Early Fusion…

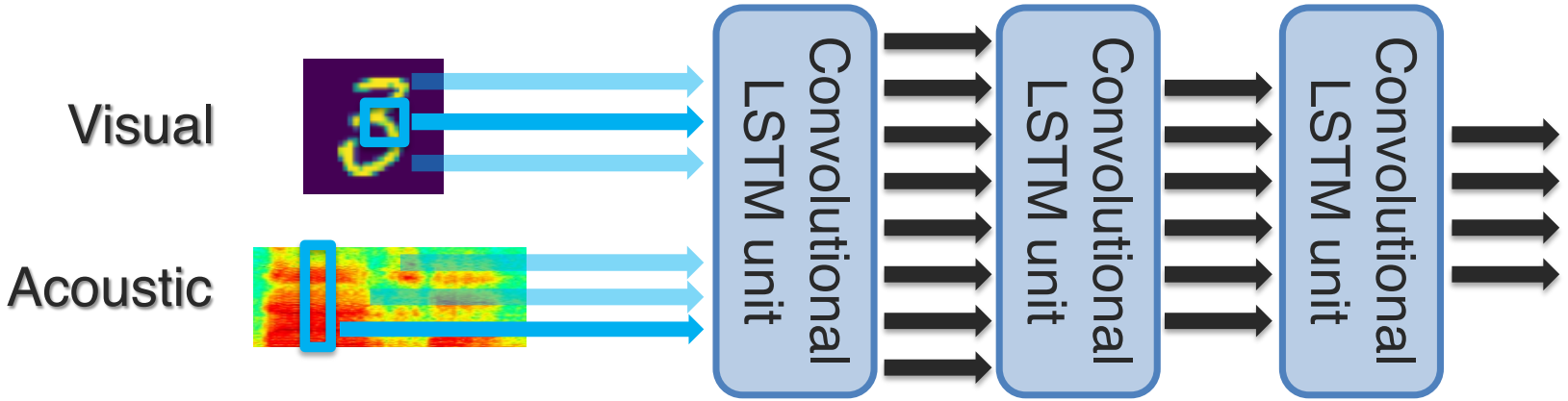


Complex Fusion



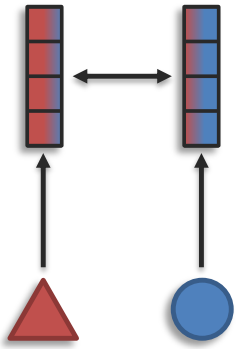
Open Challenge!

Example: From Early Fusion… to Very Early Fusion (inspired by human brain)



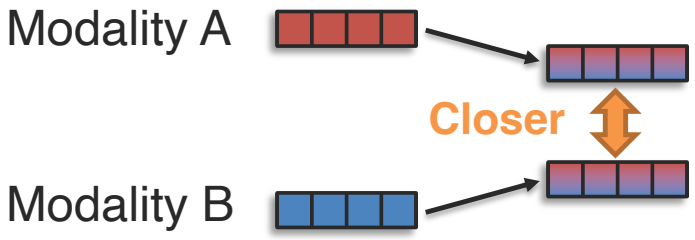
Barnum, et al. “On the Benefits of Early Fusion in Multimodal Representation Learning." *arxiv 2022*

Sub-Challenge 1b: Representation Coordination

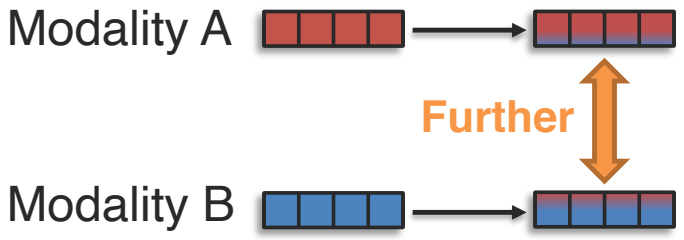


Definition: Learn multimodally-contextualized representations that are coordinated through their cross-modal interactions

Strong Coordination:

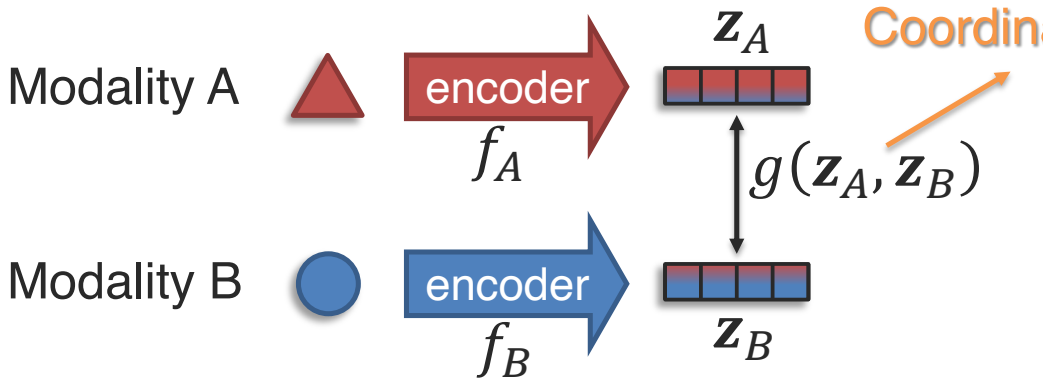


Partial Coordination:



From slides by Louis-Philippe Morency

Coordination Function



Learning with coordination function:

$$\mathcal{L} = g\big(f_A(\blacktriangle), f_B(\bullet)\big)$$

with model parameters θ_g, θ_{f_A} and θ_{f_B}

Examples of coordination function:

① Cosine similarity:

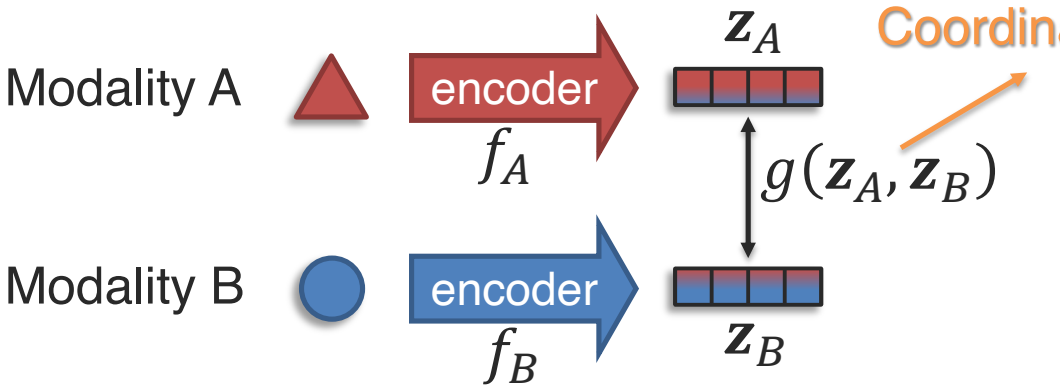
$$g(\mathbf{z}_A, \mathbf{z}_B) = \frac{\mathbf{z}_A \cdot \mathbf{z}_B}{\|\mathbf{z}_A\| \|\mathbf{z}_B\|}$$

Strong coordination!

➡ For normalized inputs (e.g., $\mathbf{z}_A - \overline{\mathbf{z}_A}$) , equivalent to *Pearson correlation coefficient*

From slides by Louis-Philippe Morency

Coordination Function



Learning with coordination function:

$$\mathcal{L} = g\big(f_A(\blacktriangle), f_B(\bullet)\big)$$

with model parameters θ_g, θ_{f_A} and θ_{f_B}

Examples of coordination function:

② Kernel similarity functions:

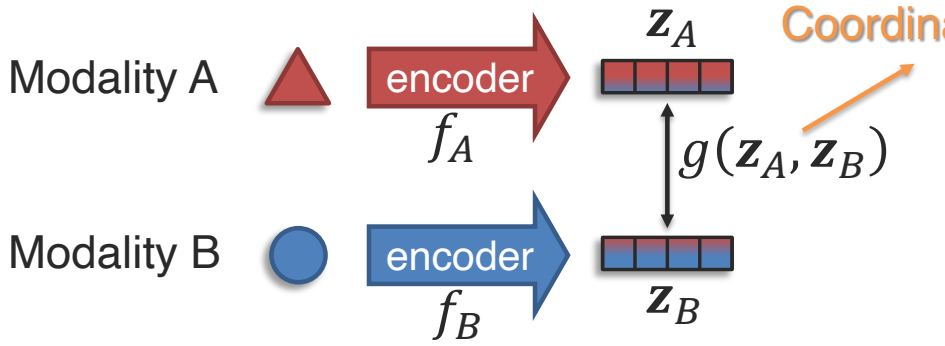
$$g(z_A, z_B) = k(z_A, z_B)$$

- Linear
- Polynomial
- Exponential
- RBF

➡ All these examples bring relatively strong coordination between modalities

From slides by Louis-Philippe Morency

Coordination Function



Learning with coordination function:

$$\mathcal{L} = g\big(f_A(\blacktriangle), f_B(\bullet)\big)$$

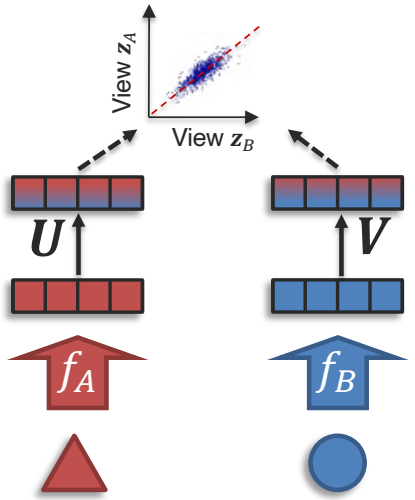
with model parameters θ_g, θ_{f_A} and θ_{f_B}

Examples of coordination function:

③ Canonical Correlation Analysis (CCA):

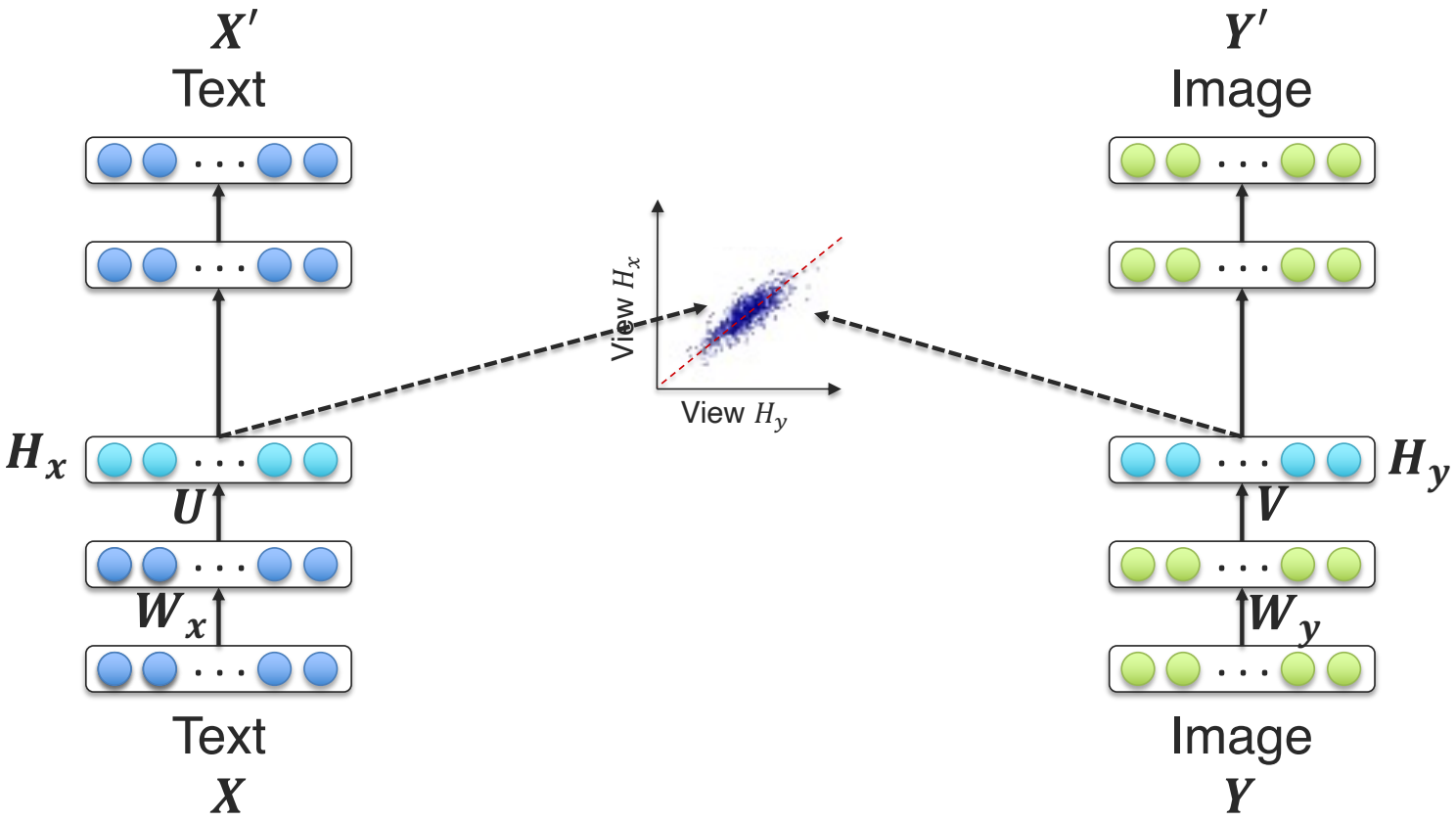
$$\underset{\boldsymbol{V},\boldsymbol{U},f_A,f_B}{\operatorname{argmax}}\ corr(\boldsymbol{z}_A, \boldsymbol{z}_B)$$

➡ CCA includes multiple projections, all orthogonal with each others



From slides by Louis-Philippe Morency

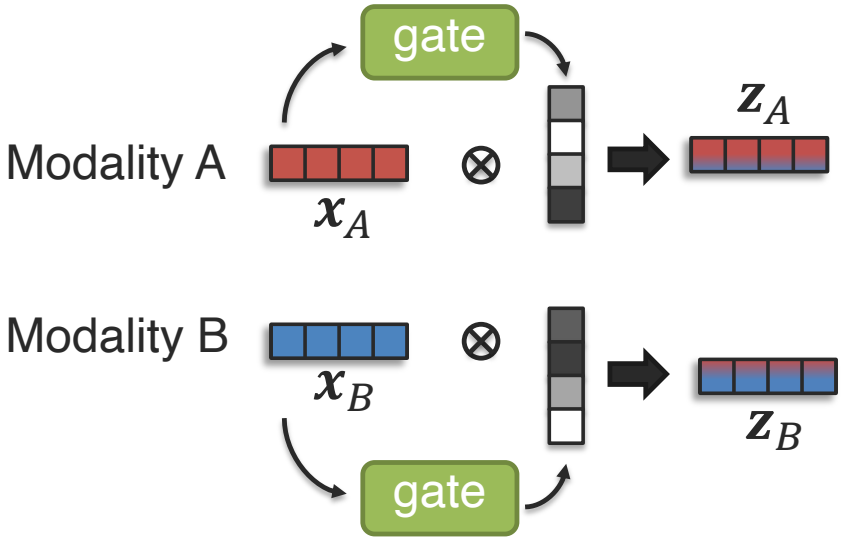
Deep Canonically Correlated Autoencoders (DCCAE)



Wang et al., On deep multi-view representation learning, PMLR 2015

From slides by Louis-Philippe Morency

Gated Coordination



Gated coordination:

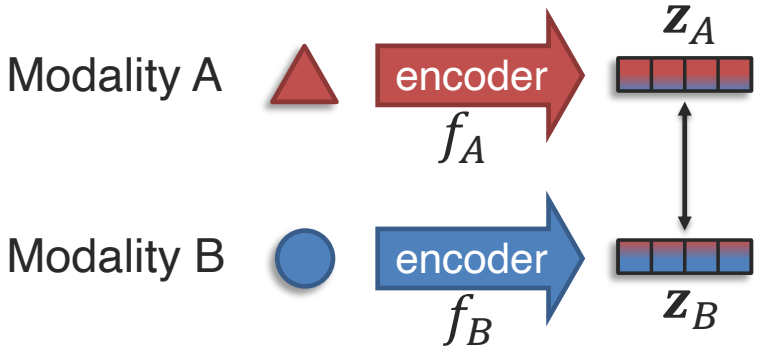
$$z_A = g_A(x_A, x_B) \cdot x_A$$

$$z_B = g_B(x_A, x_B) \cdot x_B$$

Related to attention modules in transformers

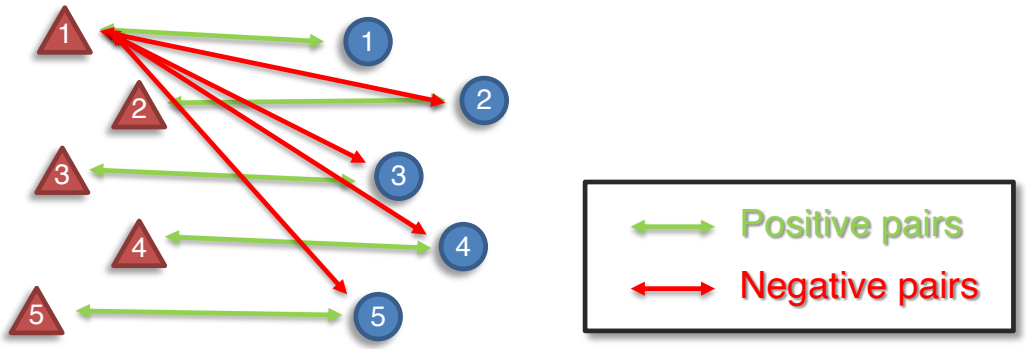
From slides by Louis-Philippe Morency

Coordination with Contrastive Learning



Paired data: $\{ \blacktriangle, \bullet \}$

(e.g., images and text descriptions)



Contrastive loss:

➡ brings positive pairs closer and pushes negative pairs apart

Simple contrastive loss:

$$\max\{0, \alpha + sim(\boldsymbol{z}_A, \boldsymbol{z}_B^+) - sim(\boldsymbol{z}_A, \boldsymbol{z}_B^-)\}$$

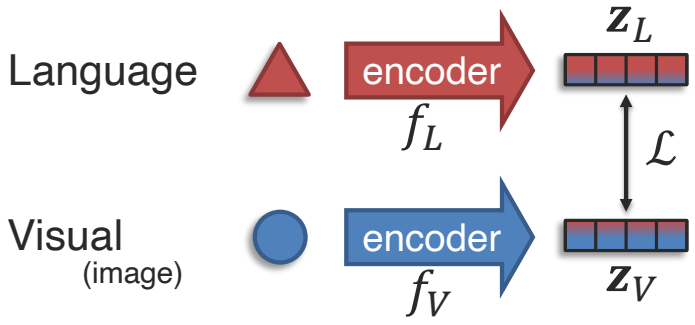
positive pairs

negative pair

Similarity functions are often cosine similarity

From slides by Louis-Philippe Morency

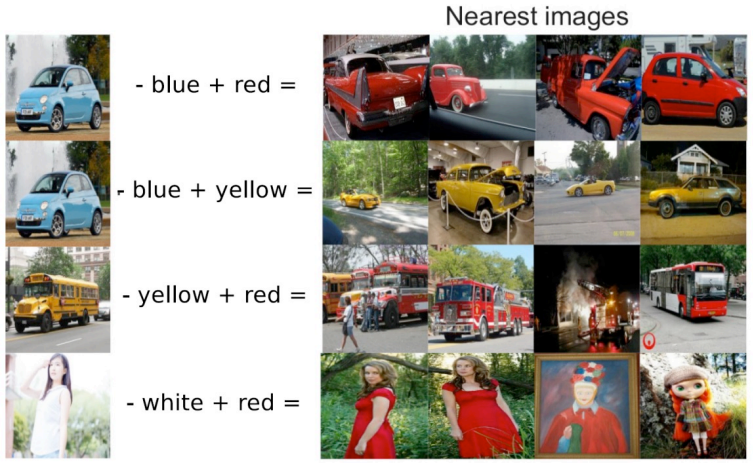
Example – Visual-Semantic Embeddings

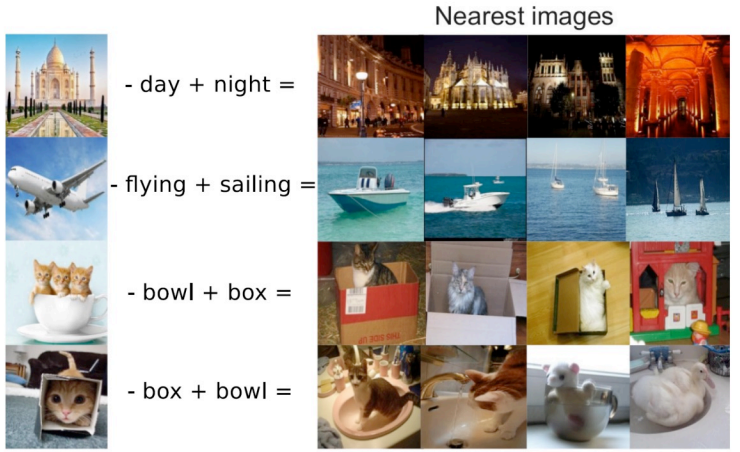


Two contrastive loss terms:

$$\max\{0, \alpha + sim(\boldsymbol{z}_L, \boldsymbol{z}_V^+) - sim(\boldsymbol{z}_L, \boldsymbol{z}_V^-)\}$$

$$+ \max\{0, \alpha + sim(\boldsymbol{z}_V, \boldsymbol{z}_L^+) - sim(\boldsymbol{z}_V, \boldsymbol{z}_L^-)\}$$

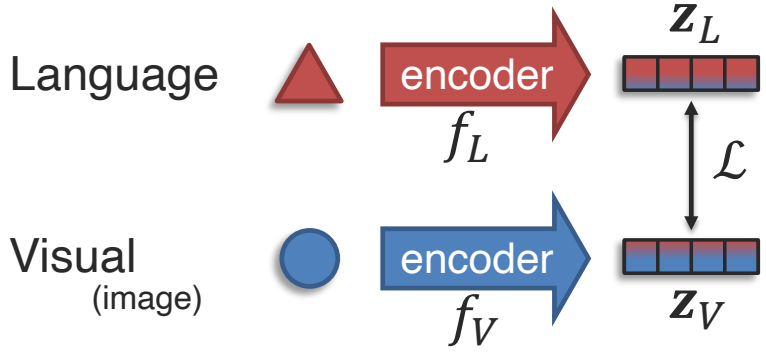




Kiros et al., Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models, NIPS 2014

From slides by Louis-Philippe Morency

Example – CLIP (Contrastive Language–Image Pre-training)



Popular contrastive loss: InfoNCE

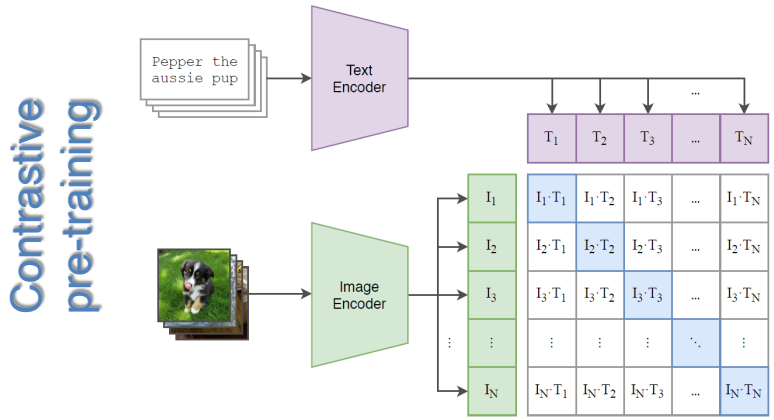
$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} log \frac{sim(z_A^i, z_B^i)}{\sum_{j=1}^{N} sim(z_A^i, z_B^j)}$$

positive pairs

negative pairs and positive pairs

Similarity function can be cosine similarity

Positive and negative pairs:



➡ f_L and f_V are great encoders for language-vision tasks

➡ z_L and z_V are coordinated but not identical representation spaces

Radford et al., Learning Transferable Visual Models From Natural Language Supervision, arxiv 2021

From slides by Louis-Philippe Morency

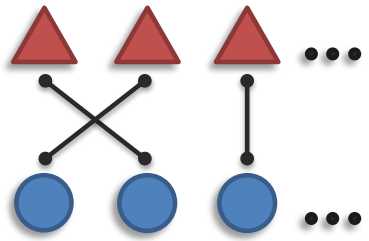
Alignment

Challenge 2: Alignment

Definition: Identifying and modeling cross-modal connections between all elements of multiple modalities, building from the data structure

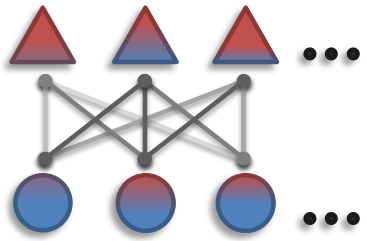
Sub-challenges:

Connections



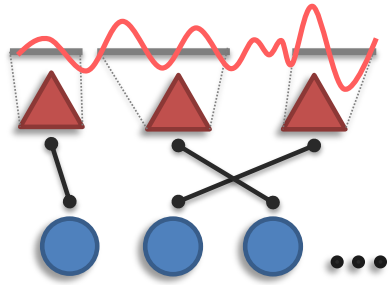
Explicit alignment
(e.g., grounding)

Aligned Representation



Implicit alignment
+ representation

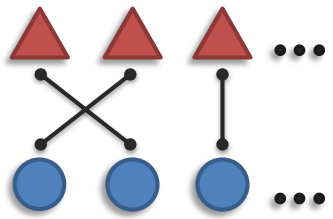
Segmentation



Granularity of
individual elements

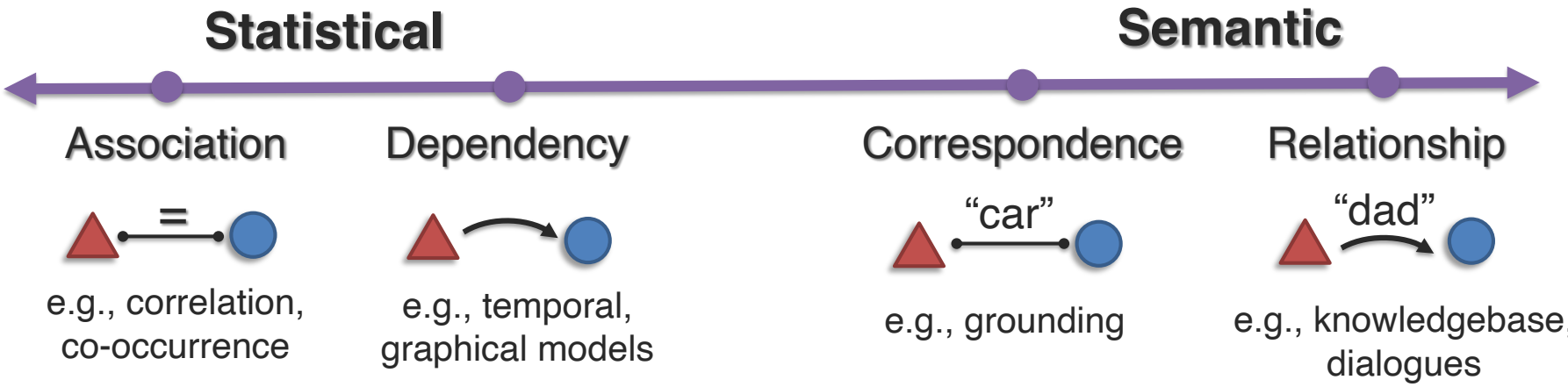
From slides by Louis-Philippe Morency

Sub-Challenge 2a: Connections



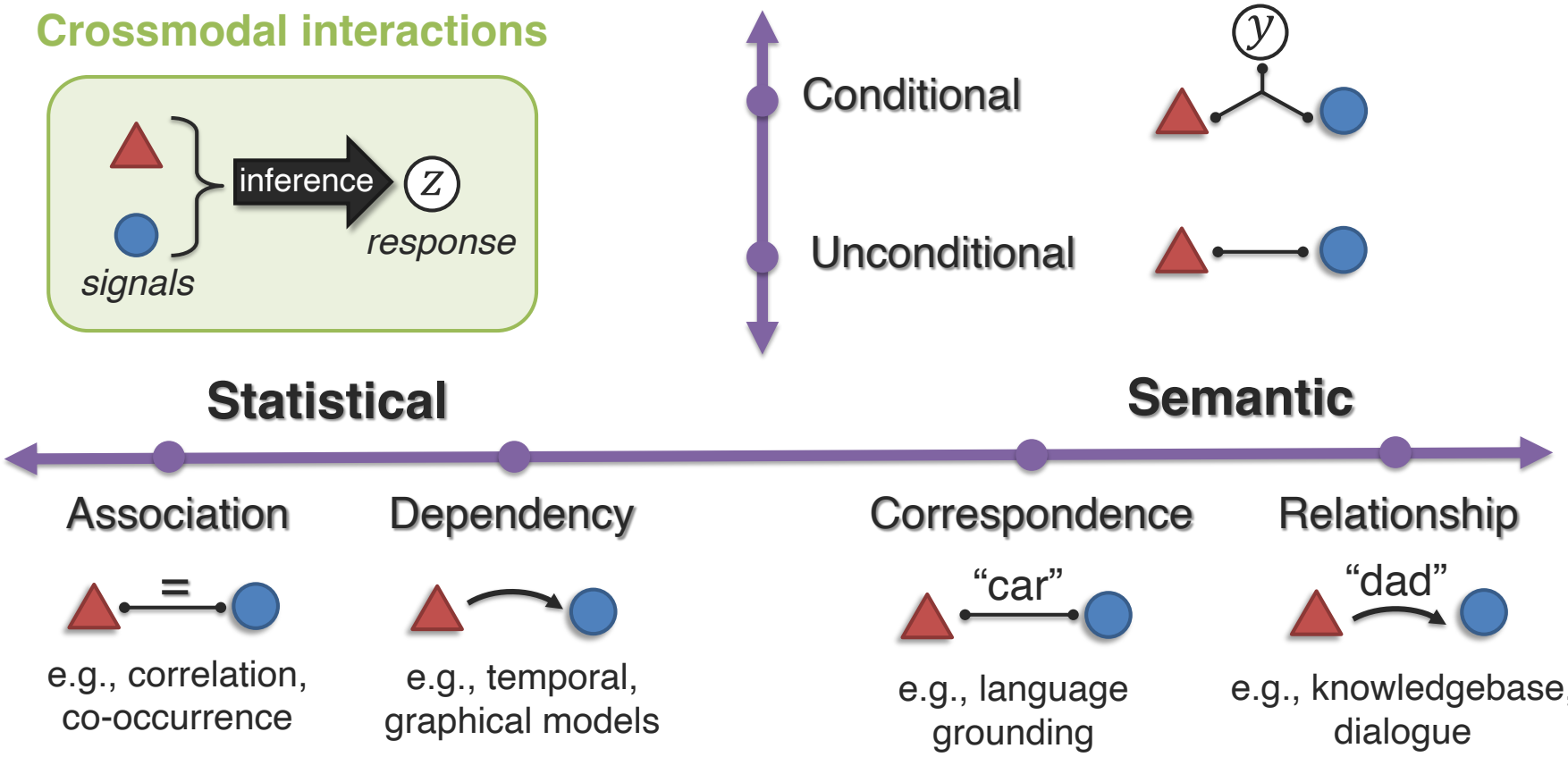
Definition: Identifying connections between elements of multiple modalities

Why should 2 elements be connected?



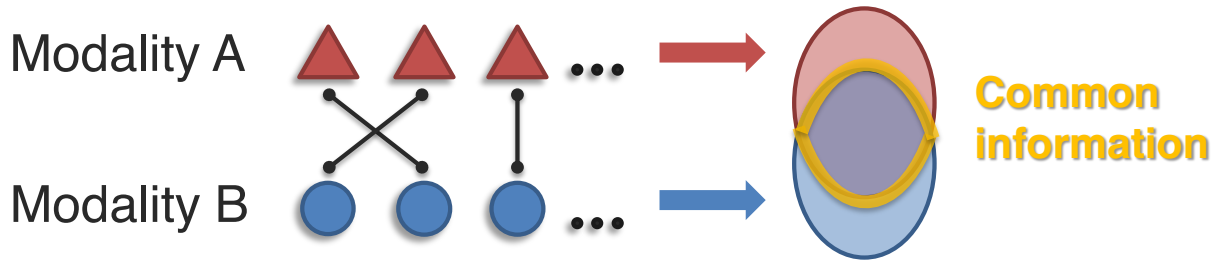
From slides by Louis-Philippe Morency

Sub-Challenge 2a: Connections



From slides by Louis-Philippe Morency

Sub-Challenge 2a: Connections



Common information

Example: language grounding

A woman reading newspaper

Statistical — **Semantic**

Association

e.g., correlation, co-occurrence

Dependency

e.g., temporal, graphical models

Correspondence

e.g., grounding

Relationship

e.g., knowledgebase, dialogue

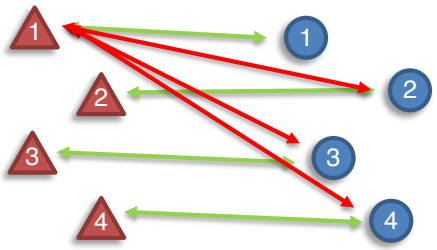
From slides by Louis-Philippe Morency

Language Grounding – Supervised Approach

Visual

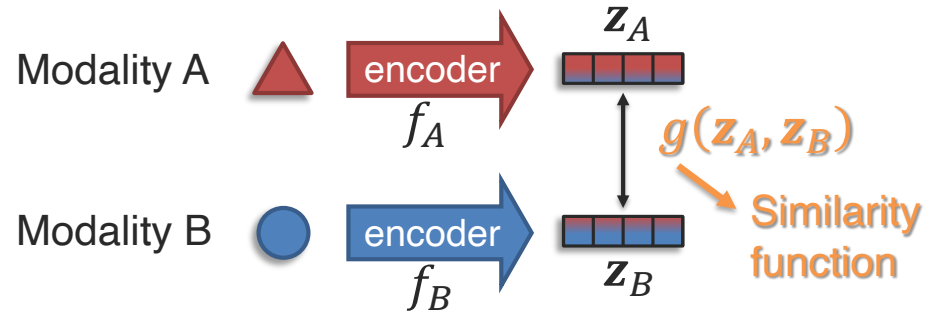
Language

Supervision: Paired data



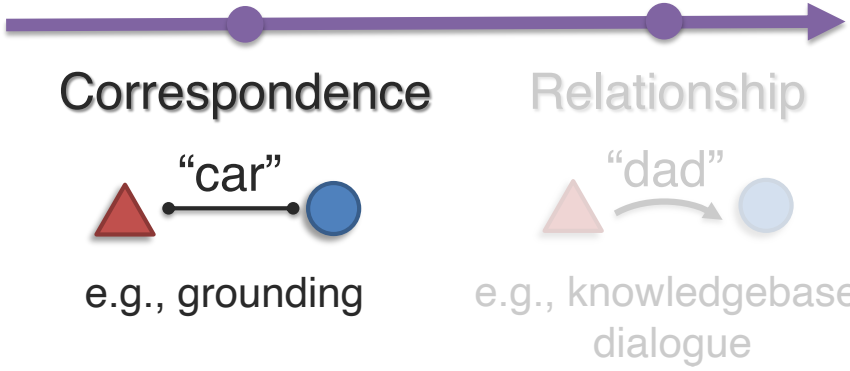
A woman reading newspaper

Learning coordinated representations:



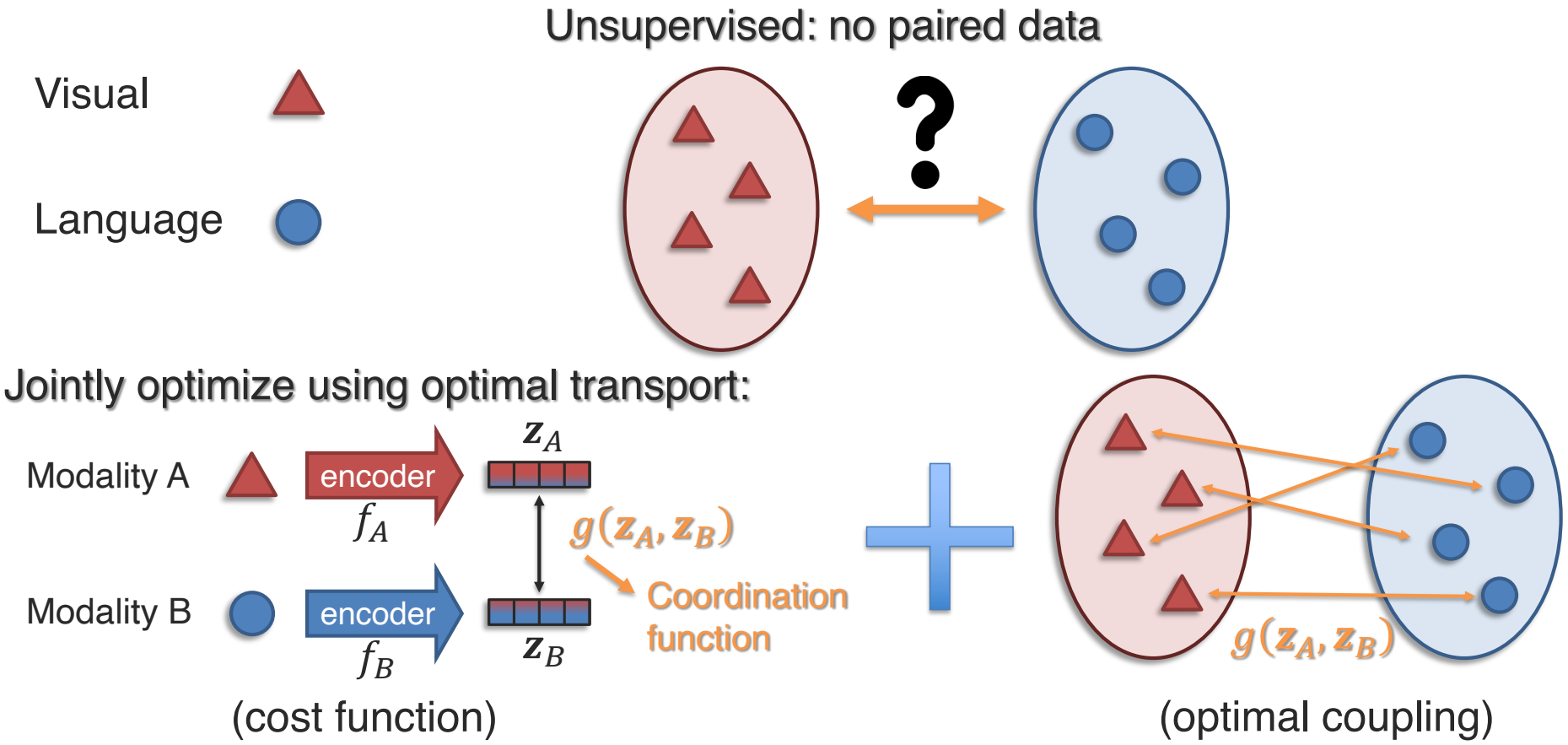
or contrastive learning

Semantic



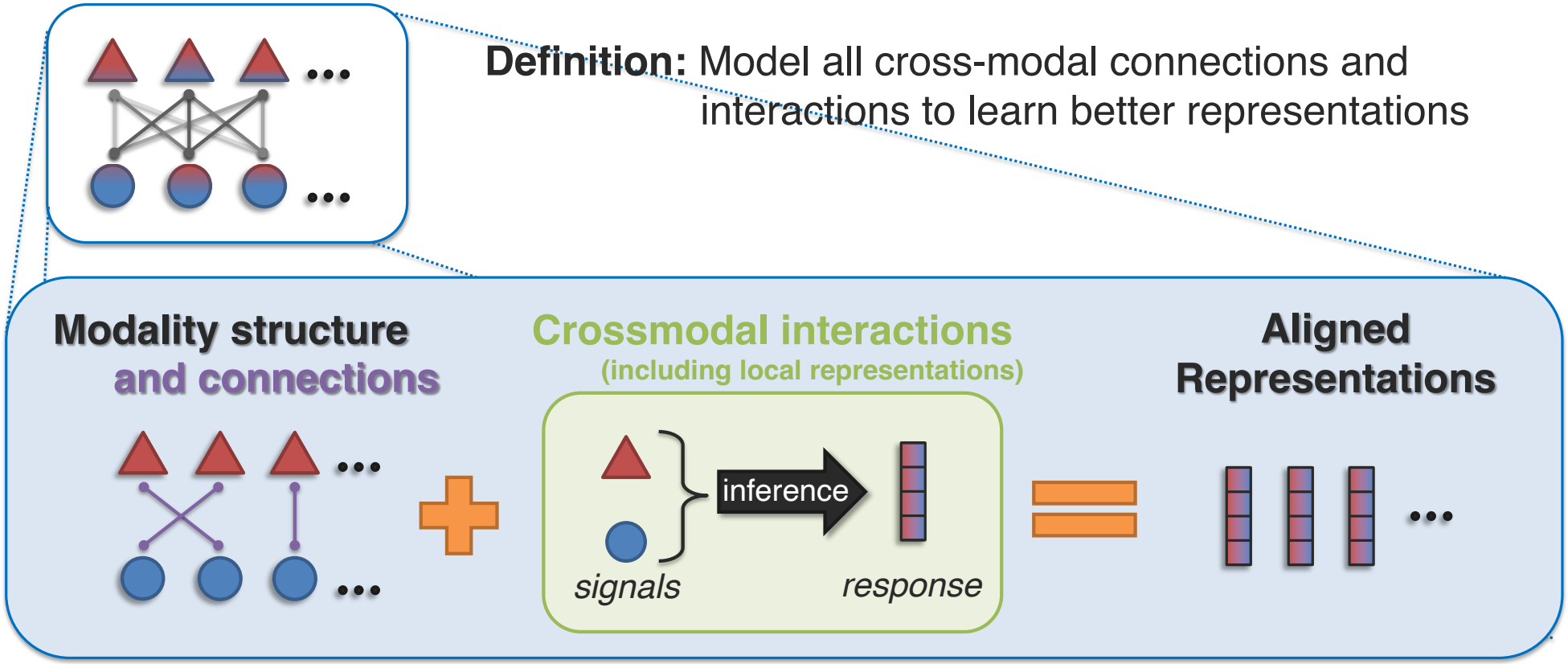
From slides by Louis-Philippe Morency

Language Grounding – Unsupervised Approach



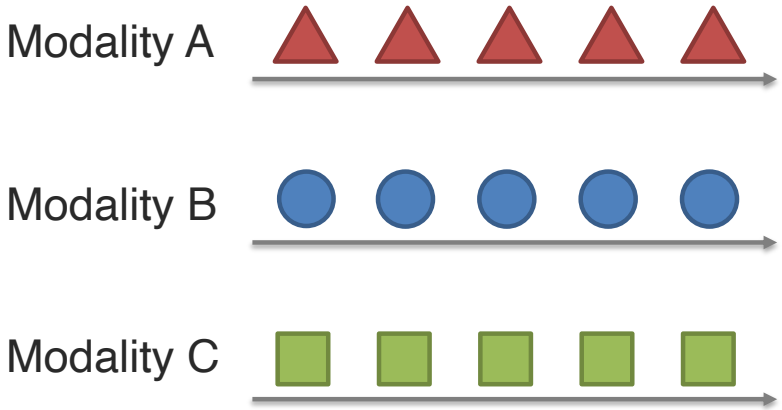
From slides by Louis-Philippe Morency

Sub-Challenge 2b: Aligned Representations



From slides by Louis-Philippe Morency

Aligned Representations – A Popular Approach



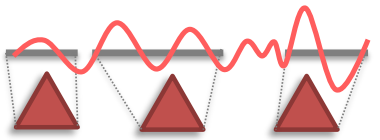
Assumptions:

① Segmented elements

objects

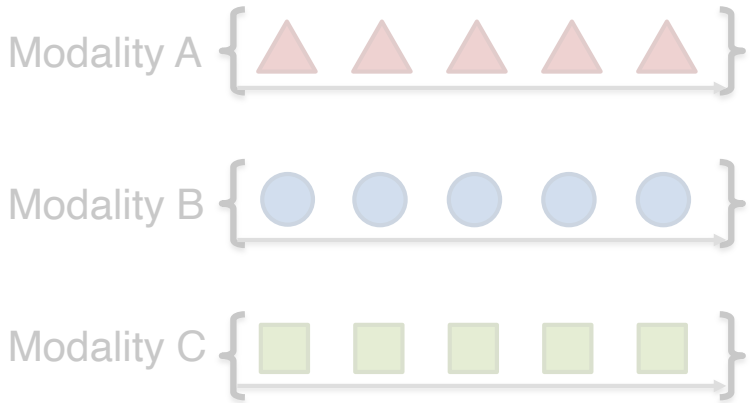
Image

➡ More about this assumption in the next sub-challenge



From slides by Louis-Philippe Morency

Aligned Representations – A Popular Approach



Assumptions:

① Segmented elements

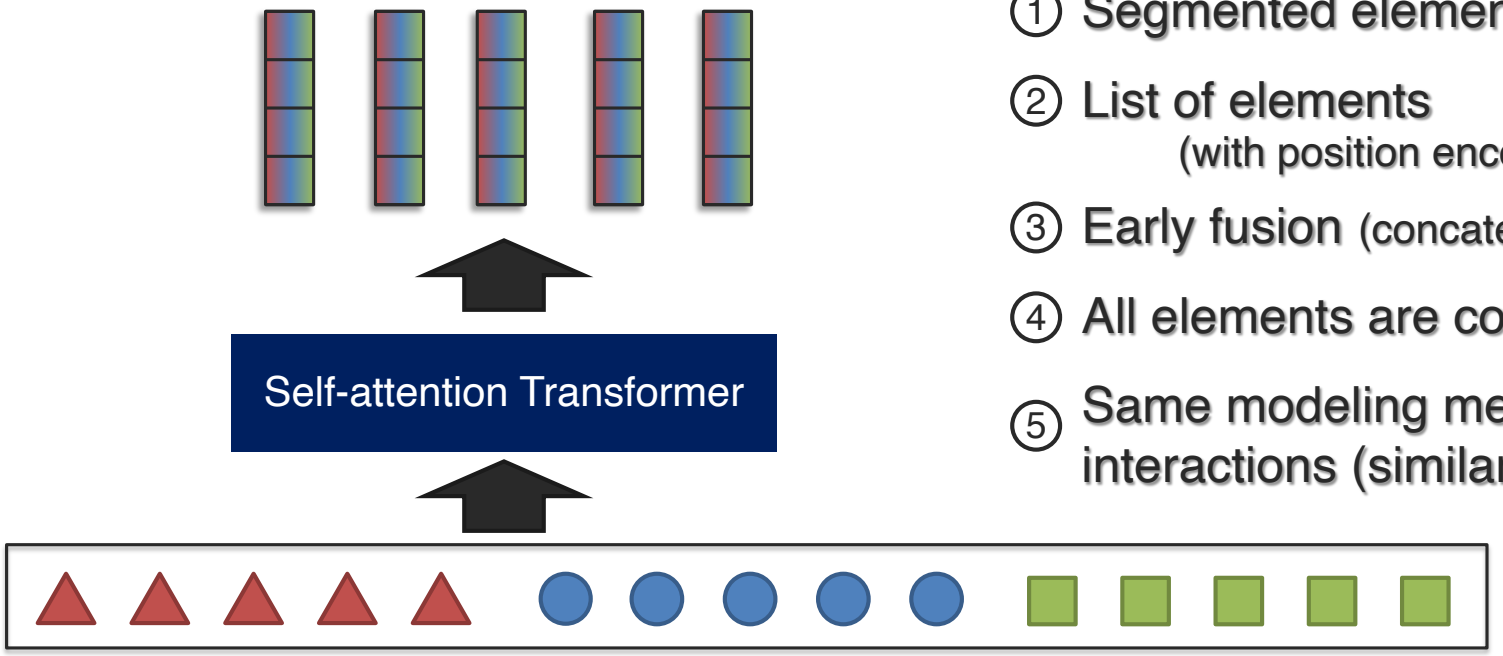
② List of elements
(with position encodings)

③ Early fusion (concatenated modalities)

Concatenate all modalities

From slides by Louis-Philippe Morency

Aligned Representations – A Popular Approach



Assumptions:

① Segmented elements

② List of elements
(with position encodings)

③ Early fusion (concatenated modalities)

④ All elements are connected

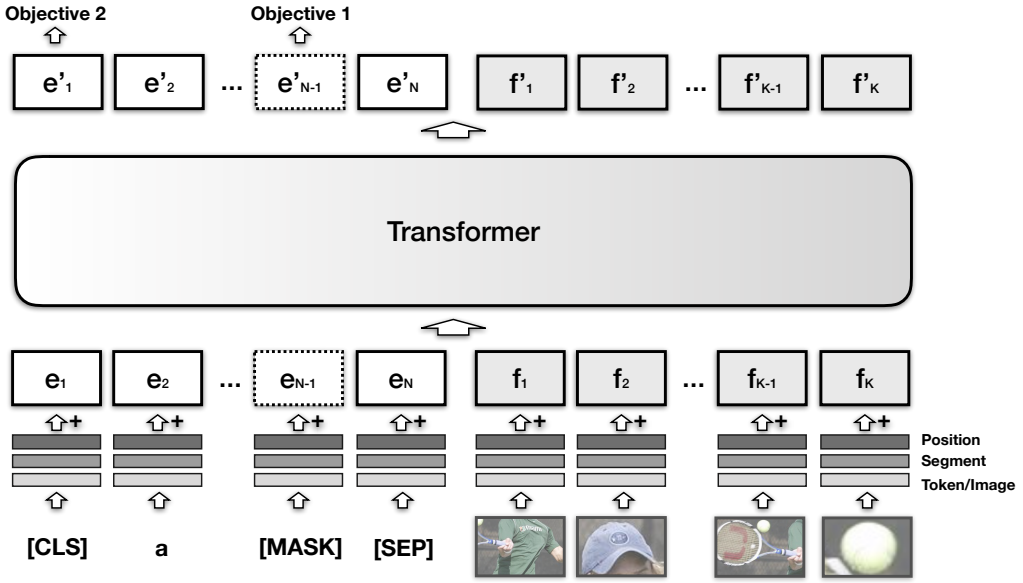
⑤ Same modeling method for all interactions (similarity kernels)

From slides by Louis-Philippe Morency

Aligned Representation – Early Fusion

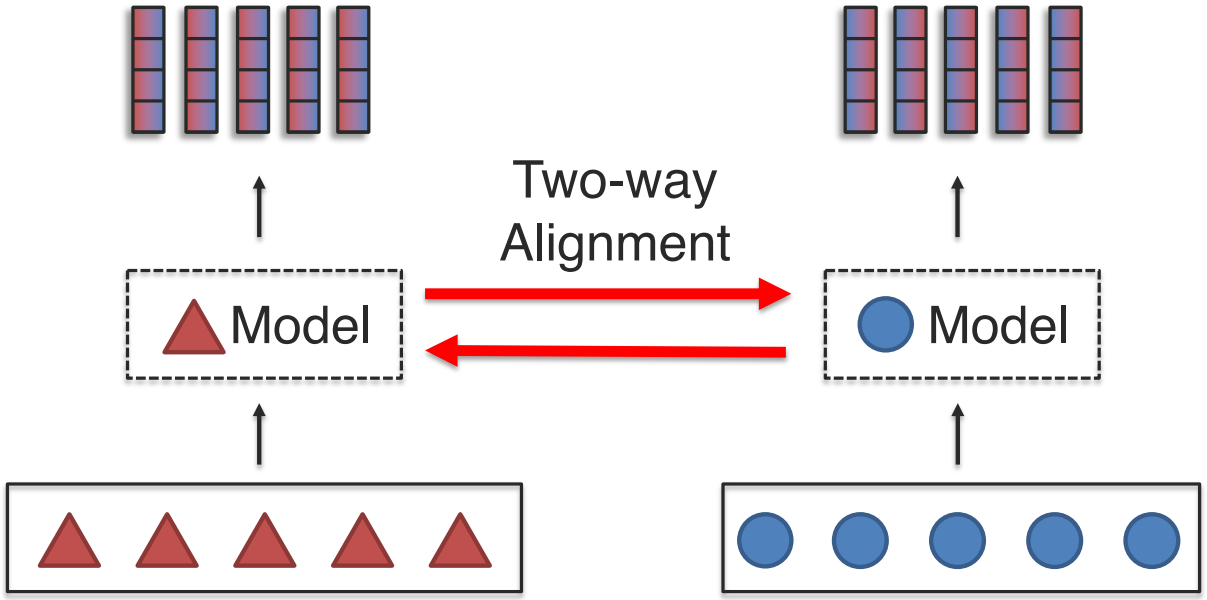
VisualBERT

A person hits a ball with a tennis racket



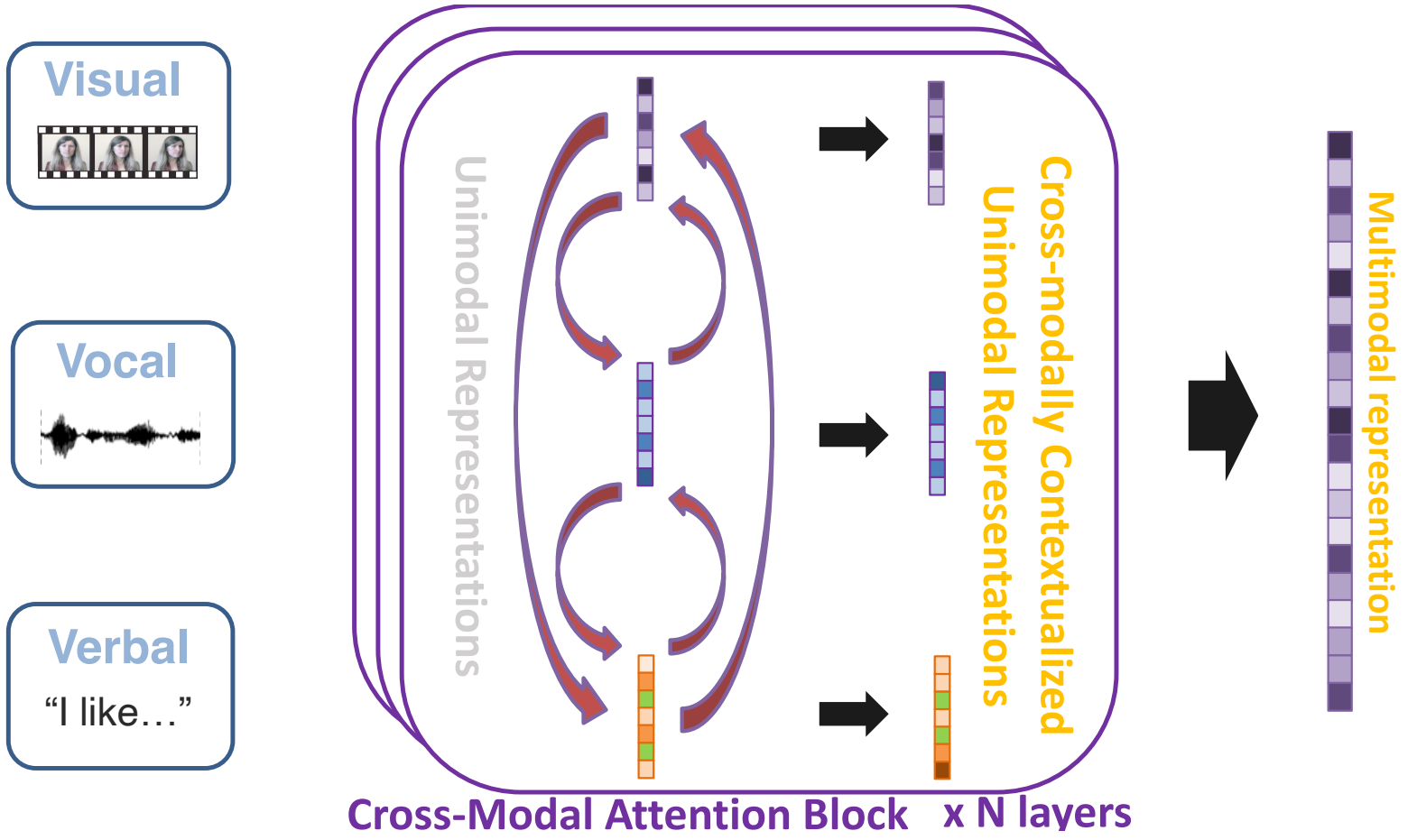
Li et al., VisualBERT: A Simple and Performant Baseline for Vision and Language, arxiv 2019

Aligned Representations – Two-Way Directional Alignment



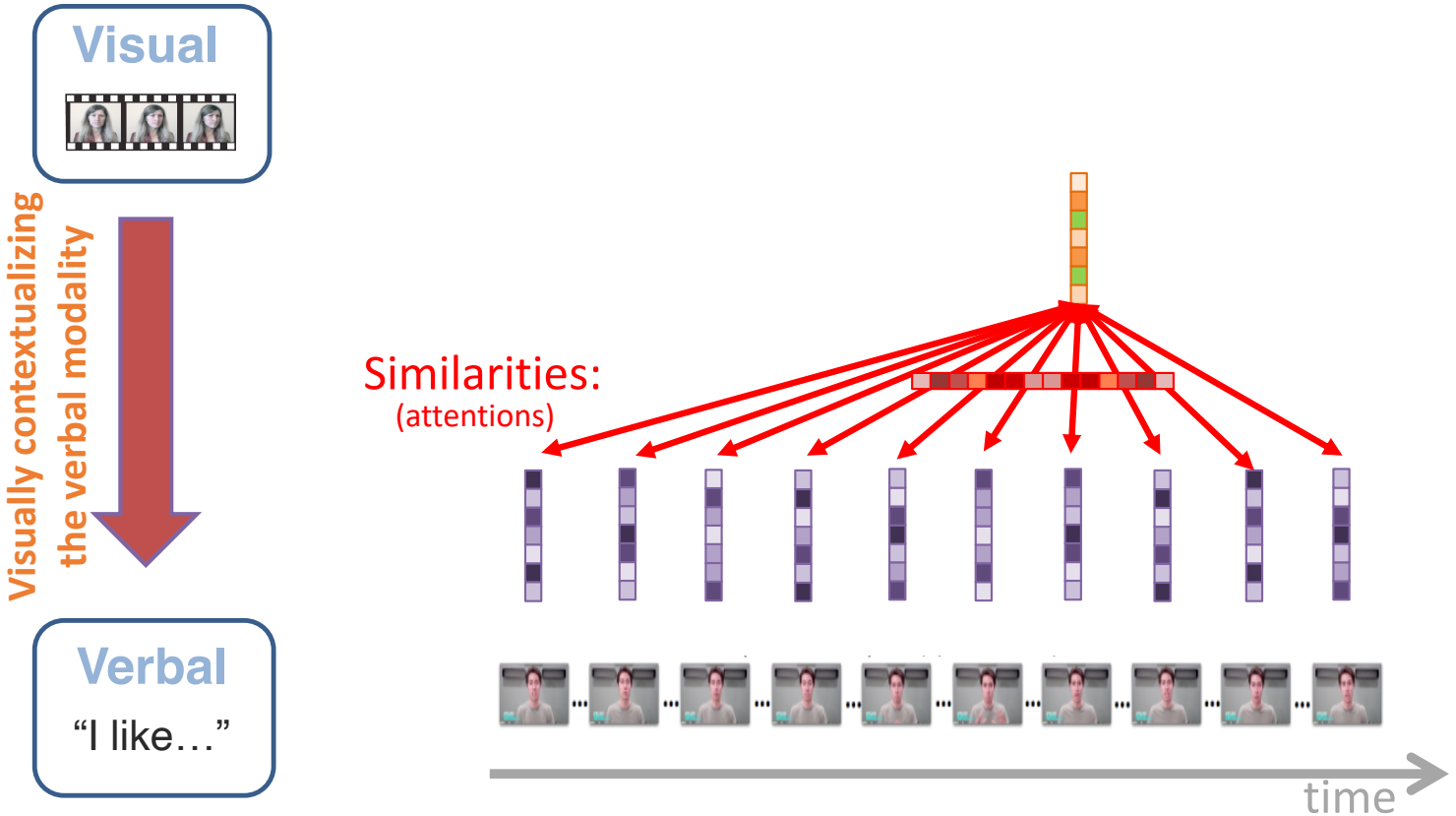
From slides by Louis-Philippe Morency

Multimodal Transformer – Pairwise Cross-Modal



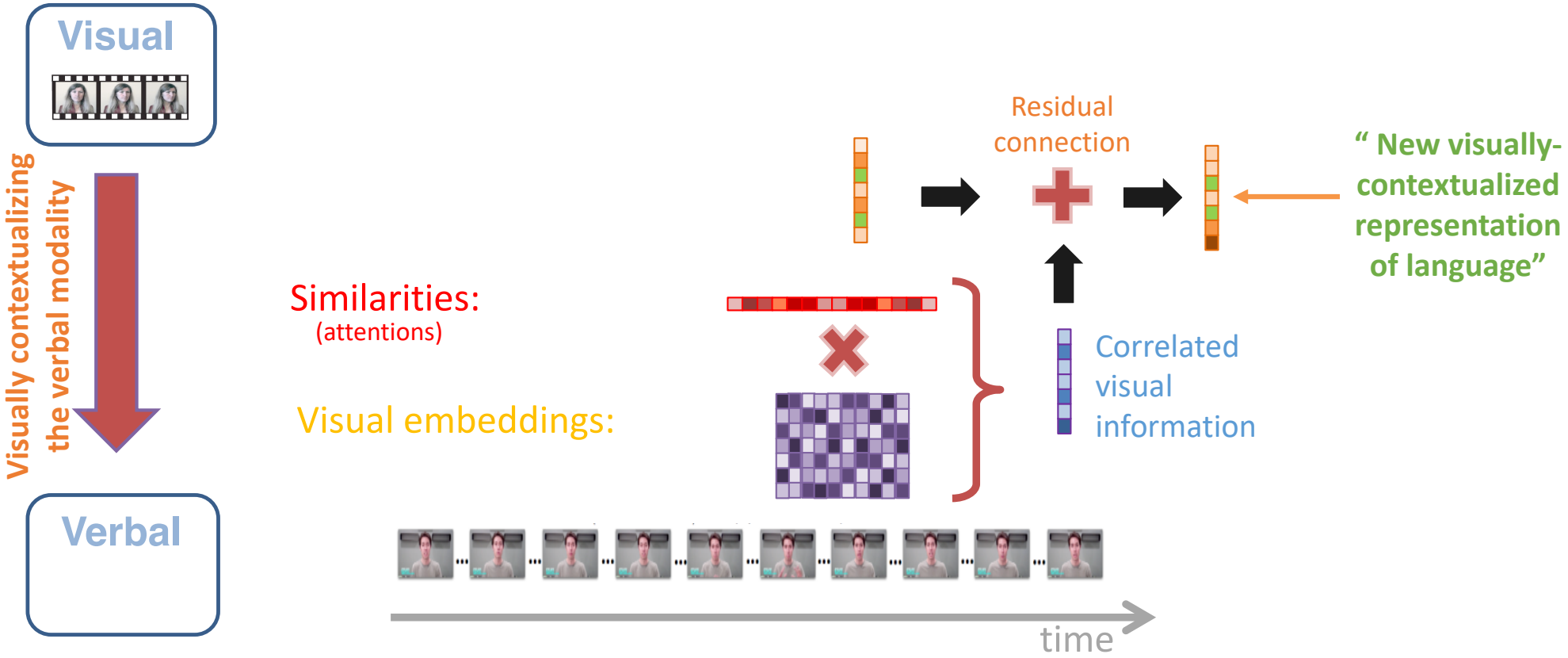
From slides by Louis-Philippe Morency

Cross-Modal Transformer Module ($\boldsymbol{V} \rightarrow \boldsymbol{L}$)



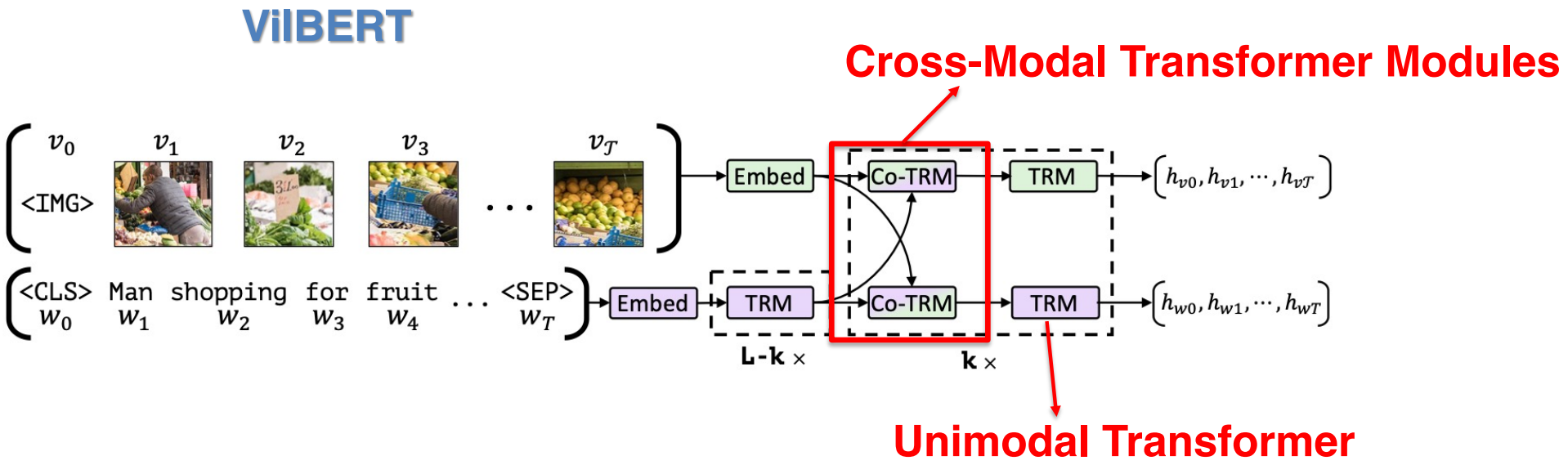
From slides by Louis-Philippe Morency

Cross-Modal Transformer Module ($V \rightarrow L$)



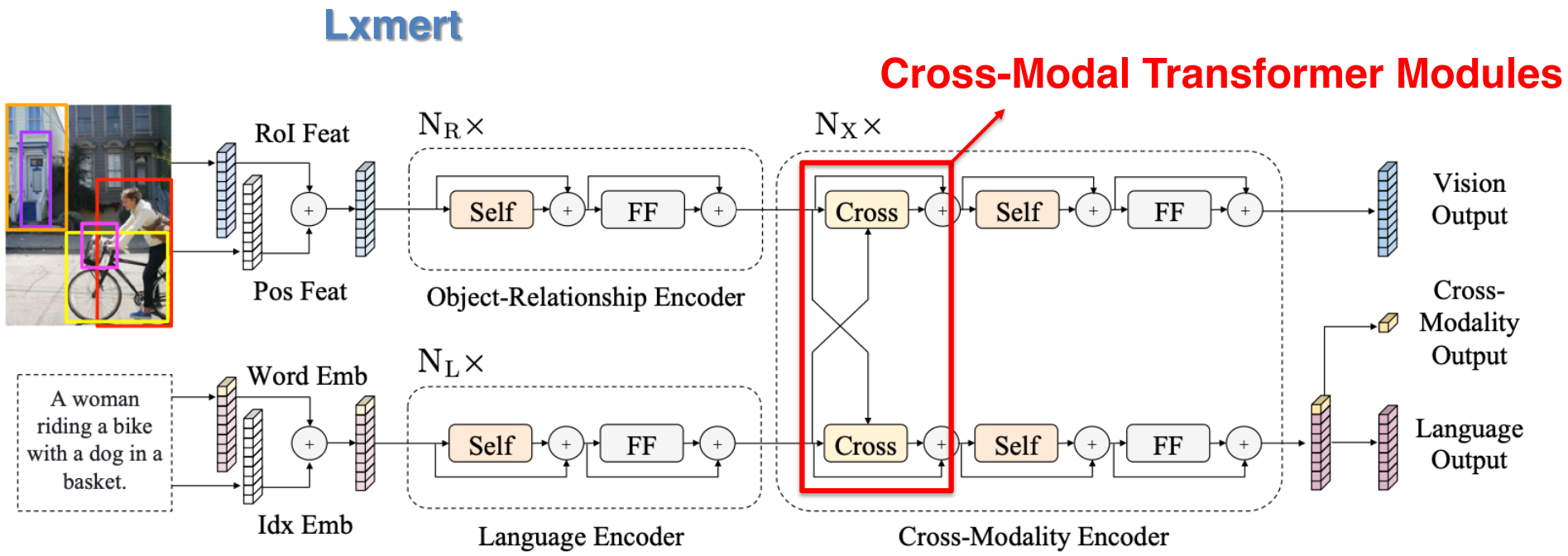
From slides by Louis-Philippe Morency

Example of Two-Way Directional Alignment



Lu, Jiasen, et al. "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks." *arXiv* (August 6, 2019).

Example of Two-Way Directional Alignment



Tan, Hao, and Mohit Bansal. "Lxmert: Learning cross-modality encoder representations from transformers." *arXiv* (August 20, 2019).