Logic-based Architectures

Neuro-symbolic modeling

THE UNIVERSITY OF UTAH

What we have seen so far: Logic to design loss functions

Most neural networks are opaque and the only interfaces we have are at the inputs and outputs

This means that most constraints will also about them

Can we write loss functions about the outputs that encourage the model to satisfy the constraints? Each constraint will be mapped to its own loss

We can then use any learning algorithm/optimizer

This lecture: Logic to design networks

Suppose we have a neural network and a constraint that involves a few nodes in the network

Can we somehow re-architect the network so that the resulting architecture (by construction) satisfies the constraint? Or almost satisfies the constraint?

Lecture outline

- Conjunctions, Disjunctions and Boolean functions as threshold networks
- The McCulloch-Pitts paper
- Knowledge-Based Artificial Neural Networks
- Augmenting neural networks with logic

Lecture outline

- Conjunctions, Disjunctions and Boolean functions as threshold networks
- The McCulloch-Pitts paper
- Knowledge-Based Artificial Neural Networks
- Augmenting neural networks with logic

Can we represent a logical statement as a neural network?

Can we represent a logical statement as a neural network?

Example 1: Consider the function

$$f_1 = X_1 \wedge X_2$$

Can we represent a logical statement as a neural network?

Example 1: Consider the function

$$f_1 = X_1 \wedge X_2$$

This function is equivalent to the linear threshold unit $X_1 + X_2 \geq 2$

Easy to verify this.

The function is true if, and only if, both the variables are set to `true`. That is, the number of `true`'s (i.e. ones) in the summation is at least two

Can we represent a logical statement as a neural network?

Example 1: Consider the function

$$f_1 = X_1 \wedge X_2$$

This function is equivalent to the linear threshold unit $X_1 + X_2 \geq 2$

And linear threshold units are one layer networks with a threshold activation

$$f_1 = \mathrm{sgn}(X_1 + X_2 - 2)$$

Can we represent a logical statement as a neural network?

Example 2: Consider the function

$$f_2 = X_1 \wedge \neg X_2$$

Can we represent a logical statement as a neural network?

Example 2: Consider the function

$$f_2 = X_1 \wedge \neg X_2$$

This function is equivalent to the linear threshold unit $X_1 + (1 - X_2) \geq 2$

That is, $X_1 - X_2 \geq 1$

Can we represent a logical statement as a neural network?

Example 2: Consider the function

$$f_2 = X_1 \land \neg X_2$$

This function is equivalent to the linear threshold unit $X_1 + (1 - X_2) \geq 2$

That is, $X_1 - X_2 \geq 1$

And linear threshold units are one layer networks with a threshold activation

$$f_2 = \text{sgn}(X_1 - X_2 - 1)$$

Can we represent a logical statement as a neural network?

Example 3: Consider the function

$$f_3 = X_1 \vee X_2 \vee X_3 \vee X_4$$

Can we represent a logical statement as a neural network?

Example 3: Consider the function

$$f_3 = X_1 \vee X_2 \vee X_3 \vee X_4$$

This function is equivalent to the linear threshold unit $X_1 + X_2 + X_3 + X_4 \geq 1$

Can we represent a logical statement as a neural network?

Example 3: Consider the function

$$f_3 = X_1 \vee X_2 \vee X_3 \vee X_4$$

This function is equivalent to the linear threshold unit $X_1 + X_2 + X_3 + X_4 \geq 1$

And linear threshold units are one layer networks with a threshold activation

$$f_3 = \mathrm{sgn}\left(\sum X_i - 1\right)$$

Can we represent a logical statement as a neural network?

Example 4: Consider the function

$$f_4 = X_1 \to X_2 \equiv \neg X_1 \vee X_2$$

Can we represent a logical statement as a neural network?

Example 4: Consider the function

$$f_4 = X_1 \rightarrow X_2 \equiv \neg X_1 \vee X_2$$

This function is equivalent to the linear threshold unit $(1 - X_1) + X_2 \geq 1$

That is, $-X_1 + X_2 \geq 0$

Can we represent a logical statement as a neural network?

Example 4: Consider the function

$$f_4 = X_1 \to X_2 \equiv \neg X_1 \vee X_2$$

This function is equivalent to the linear threshold unit $(1 - X_1) + X_2 \geq 1$

That is, $-X_1 + X_2 \geq 0$

And linear threshold units are one layer networks with a threshold activation

$$f_4 = \text{sgn}(-X_1 + X_2)$$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \vee l_2 \vee \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \vee l_2 \vee \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq 1$$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \vee l_2 \vee \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq 1$$

But some literals show up with a negation. For such literals $l_i = 1 - X_i$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \vee l_2 \vee \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq 1$$

But some literals show up with a negation. For such literals $l_i = 1 - X_i$

$$\mathrm{sgn}\left(\sum_{i \in P} X_i - \sum_{i \in N} X_i + |N| - 1\right)$$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \vee l_2 \vee \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq 1$$

But some literals show up with a negation. For such literals $l_i = 1 - X_i$

$$\mathrm{sgn}\left(\sum_{i \in P} X_i - \sum_{i \in N} X_i + |N| - 1\right)$$

Literals that have positive polarity (i.e., not negated)

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \vee l_2 \vee \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq 1$$

But some literals show up with a negation. For such literals $l_i = 1 - X_i$

$$\operatorname{sgn}\left(\sum_{i \in P} X_i - \sum_{i \in N} X_i + |N| - 1\right)$$

Literals that have positive polarity (i.e., not negated)

Literals that have positive polarity (i.e., not negated)

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \wedge l_2 \wedge \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \wedge l_2 \wedge \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq \text{number of variables}$$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \wedge l_2 \wedge \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq \text{number of variables}$$

But some literals show up with a negation. For such literals $l_i = 1 - X_i$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \wedge l_2 \wedge \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq \text{number of variables}$$

But some literals show up with a negation. For such literals $l_i = 1 - X_i$

$$\text{sgn}\left(\sum_{i \in P} X_i - \sum_{i \in N} X_i - |P|\right)$$

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \wedge l_2 \wedge \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq \text{number of variables}$$

But some literals show up with a negation. For such literals $l_i = 1 - X_i$

$$\operatorname{sgn}\left(\sum_{i \in P} X_i - \sum_{i \in N} X_i - |P|\right)$$

Literals that have positive polarity (i.e., not negated)

Conjunctions & disjunctions are linearly separable

This offers a simple recipe to write them as threshold linear units

Suppose $f = l_1 \wedge l_2 \wedge \cdots$,

where l_i is the variable X_i or its negation $\neg X_i$

The expression f is true if the number of true literals is at least 1. That is, if

$$\sum_i l_i \geq \text{number of variables}$$

But some literals show up with a negation. For such literals $l_i = 1 - X_i$

$$\text{sgn}\left(\sum_{i \in P} X_i - \sum_{i \in N} X_i - |P|\right)$$

Literals that have positive polarity (i.e., not negated)

Literals that have positive polarity (i.e., not negated)

How can we extend this to arbitrary Boolean functions?

Ideas?

How can we extend this to arbitrary Boolean functions?

Any Boolean function can be written in as a conjunctive normal form

A conjunctive normal form is a conjunction of disjunctions

- We know how to write each disjunction as a one layer network

- Each disjunction produces a 0 or a 1

- The final function is a conjunction of these disjunction values. We know how to write the conjunction as a one layer network that operates on top of the disjunctions

Let’s see an example

Arbitrary Boolean function as a two-layer network

An example

Consider $f = X_1 \to (X_2 \to X_3)$

Arbitrary Boolean function as a two-layer network

An example

Consider $f = X_1 \rightarrow (X_2 \rightarrow X_3)$

In CNF $(X_1 \vee X_3) \wedge (\neg X_1 \vee X_3)$

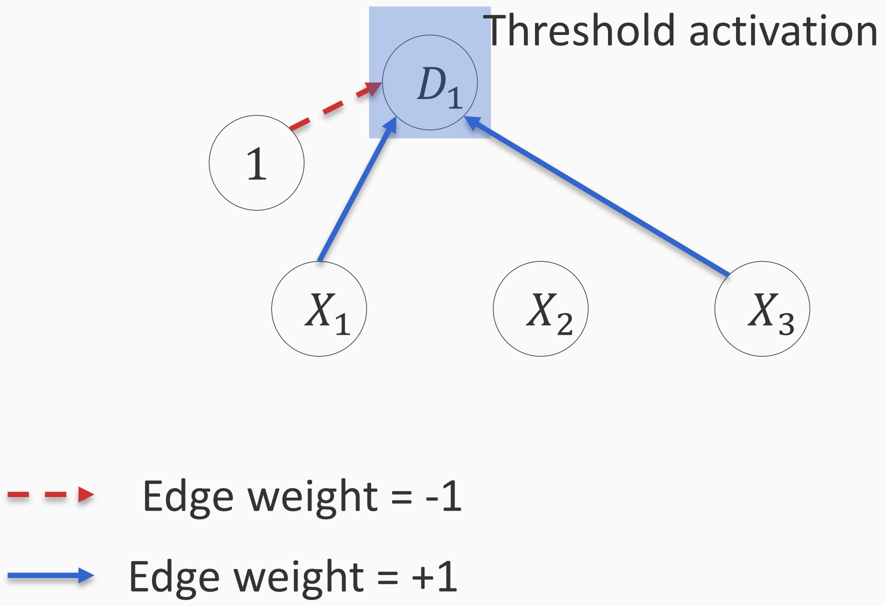
Arbitrary Boolean function as a two-layer network

An example

Consider $f = X_1 \to (X_2 \to X_3)$

In CNF $(X_1 \vee X_3) \wedge (\neg X_1 \vee X_3)$

$(X_1 \vee X_3) \equiv \text{sgn}(X_1 + X_3 - 1)$



Arbitrary Boolean function as a two-layer network

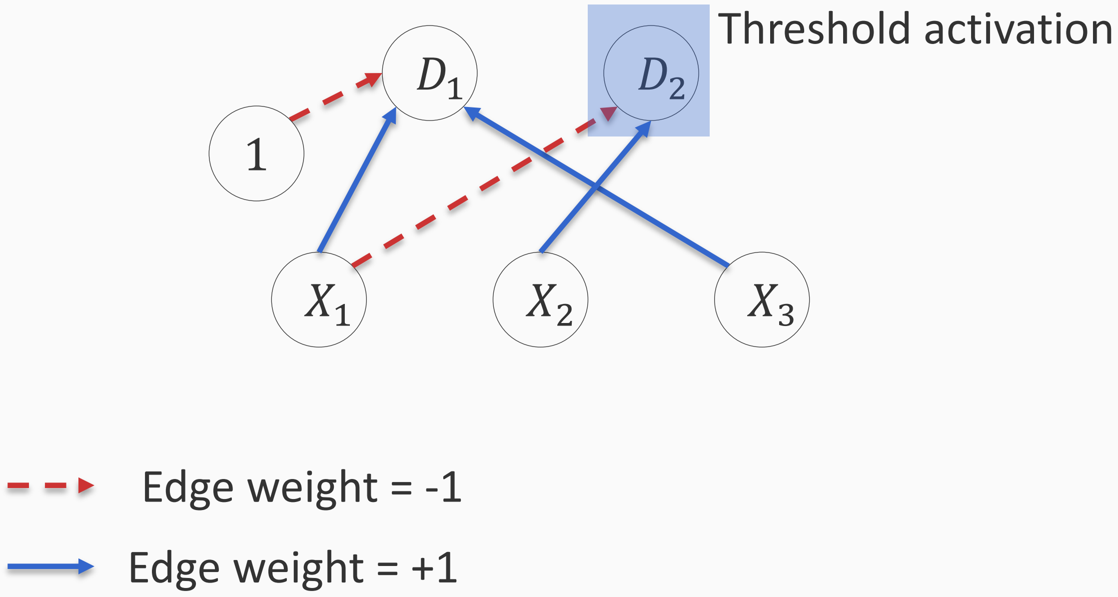
An example

Consider $f = X_1 \rightarrow (X_2 \rightarrow X_3)$

In CNF $(X_1 \vee X_3) \wedge (\neg X_1 \vee X_3)$

$(X_1 \vee X_3) \equiv \mathrm{sgn}(X_1 + X_3 - 1)$

$(\neg X_1 \vee X_2) \equiv \mathrm{sgn}(-X_1 + X_2)$



Arbitrary Boolean function as a two-layer network

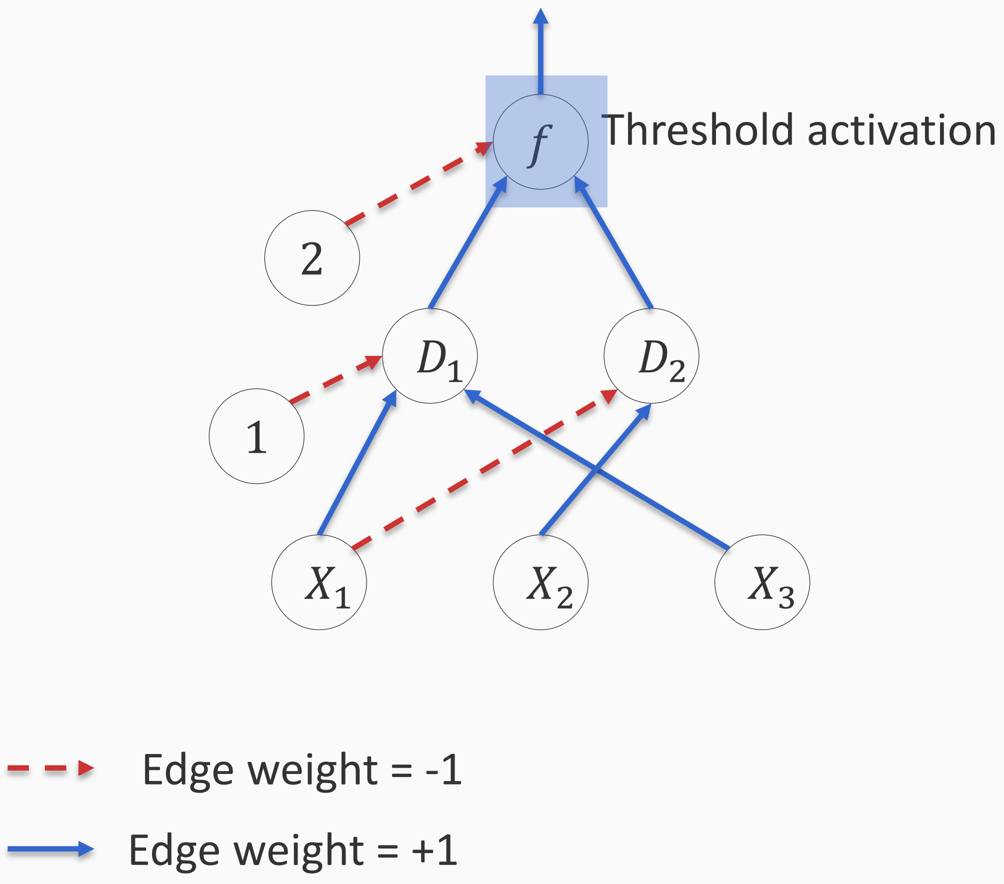
An example

Consider $f = X_1 \rightarrow (X_2 \rightarrow X_3)$

In CNF $(X_1 \vee X_3) \wedge (\neg X_1 \vee X_3)$

$$(X_1 \vee X_3) \equiv \text{sgn}(X_1 + X_3 - 1)$$

$$(\neg X_1 \vee X_2) \equiv \text{sgn}(-X_1 + X_2)$$



Exercises

1. Threshold activations produce -1 or 1, but the construction we saw treats true and false as 1 and 0 respectively. Adapt the approach for -1 and +1

2. How will this construction change for a *disjunctive normal form*?

3. If any Boolean function can be represented as two layer network, what is the catch?

Lecture outline

- Conjunctions, Disjunctions and Boolean functions as threshold networks
- The McCulloch-Pitts paper
- Knowledge-Based Artificial Neural Networks
- Augmenting neural networks with logic

The first paper to introduce artificial neural networks

BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE
IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

But wait there's more…

"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

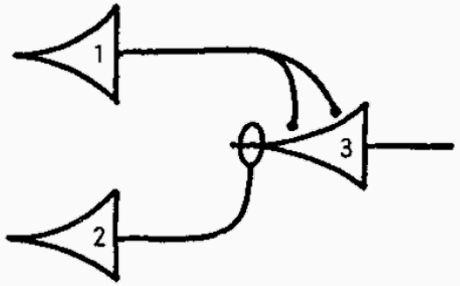
The paper shows how to construct neural networks for any Boolean function

But wait there's more…

"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:

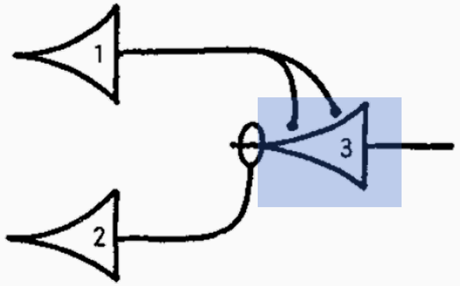


But wait there's more…

"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



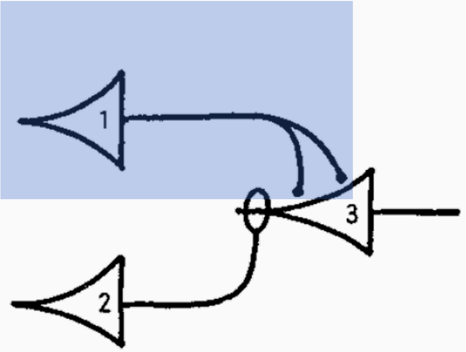
$N_3(t) = N_1(t-1) \wedge \neg N_2(t-1)$

But wait there's more…

"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



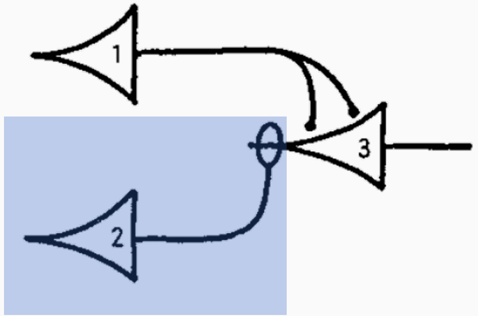
$N_3(t) = N_1(t-1) \wedge \neg N_2(t-1)$

But wait there's more…

"The method [...] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



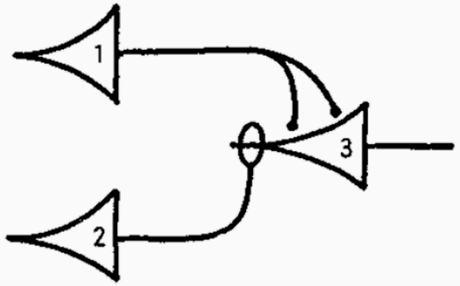
$$N_3(t) = N_1(t-1) \land \neg N_2(t-1)$$

But wait there's more…

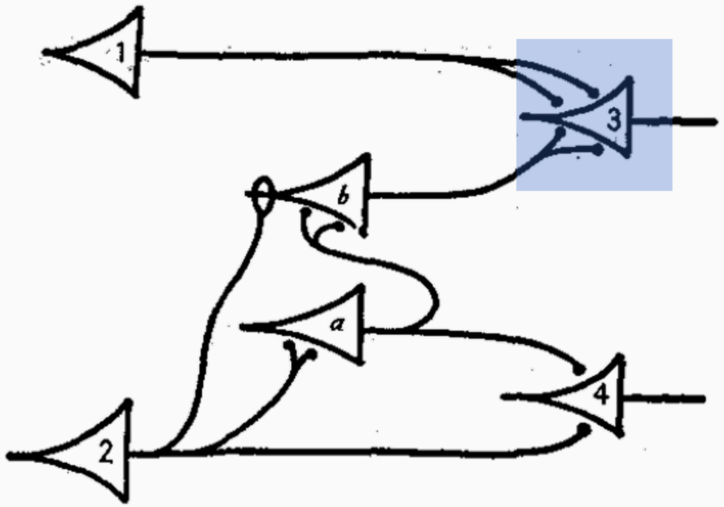
"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



$$N_3(t) = N_1(t-1) \land \neg N_2(t-1)$$



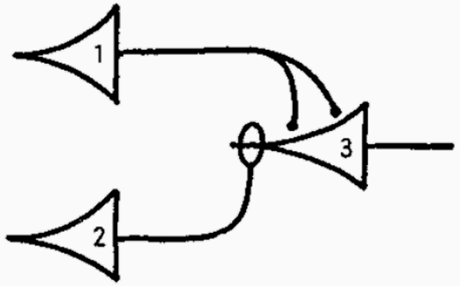
$$N_3(t) = N_1(t-1) \lor \left(\neg N_2(t-2) \land N_2(t-3)\right)$$

But wait there's more…

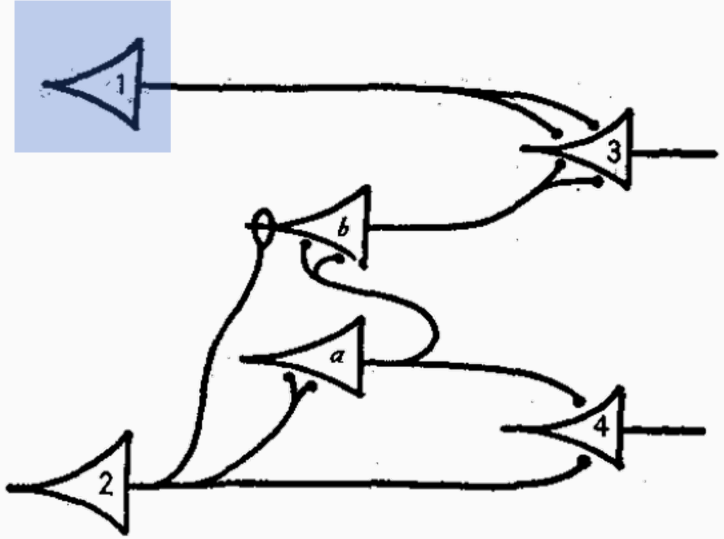
"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



$$N_3(t) = N_1(t-1) \wedge \neg N_2(t-1)$$



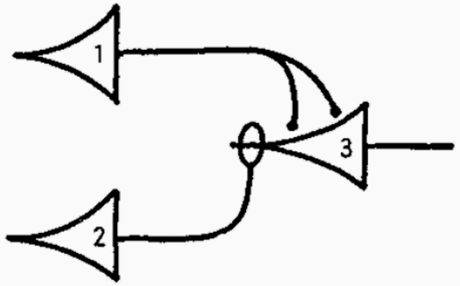
$$N_3(t) = N_1(t-1) \vee \left(\neg N_2(t-2) \wedge N_2(t-3)\right)$$

But wait there's more…

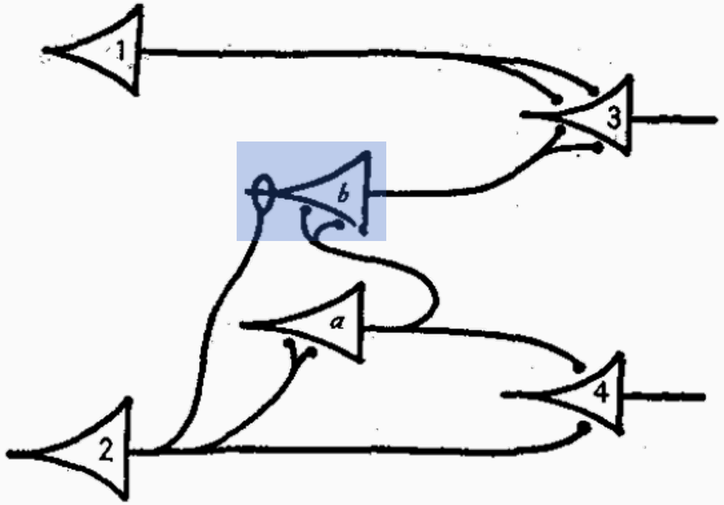
"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



$$N_3(t) = N_1(t-1) \wedge \neg N_2(t-1)$$



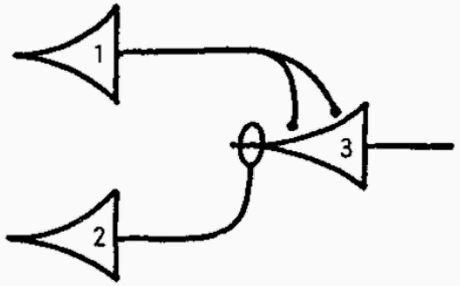
$$N_3(t) = N_1(t-1) \vee \left(\neg N_2(t-2) \wedge N_2(t-3)\right)$$

But wait there's more…

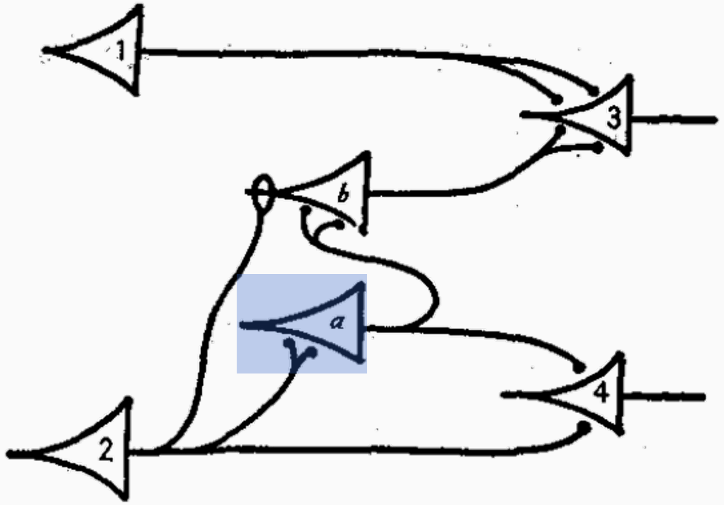
"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



$N_3(t) = N_1(t-1) \wedge \neg N_2(t-1)$



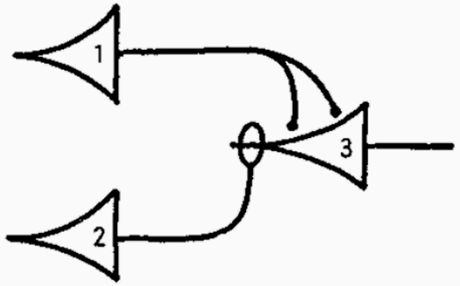
$N_3(t) = N_1(t-1) \vee \left(\neg N_2(t-2) \wedge N_2(t-3)\right)$

But wait there's more…

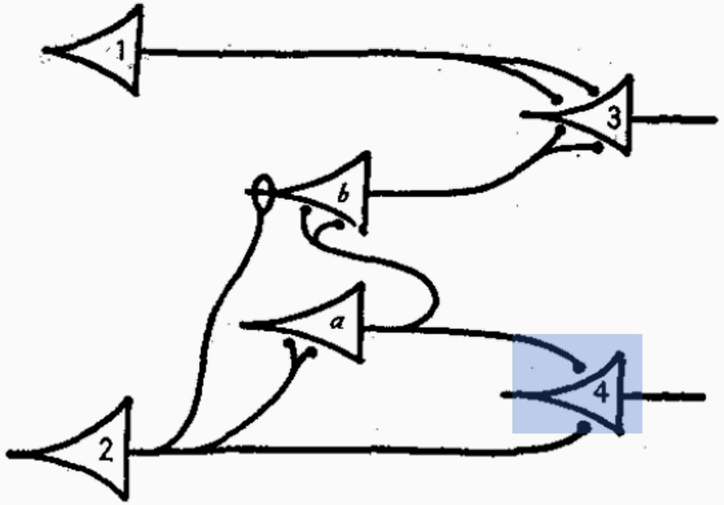
"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



$N_3(t) = N_1(t-1) \wedge \neg N_2(t-1)$



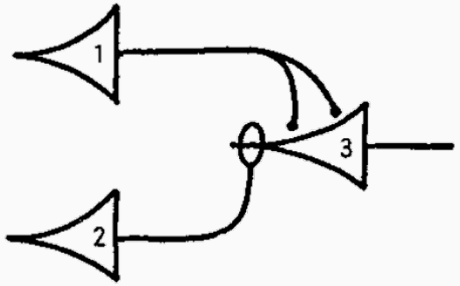
$N_3(t) = N_1(t-1) \vee \left(\neg N_2(t-2) \wedge N_2(t-3)\right)$
$N_4(t) = N_2(t-1) \wedge N_2(t-2)$

But wait there's more…

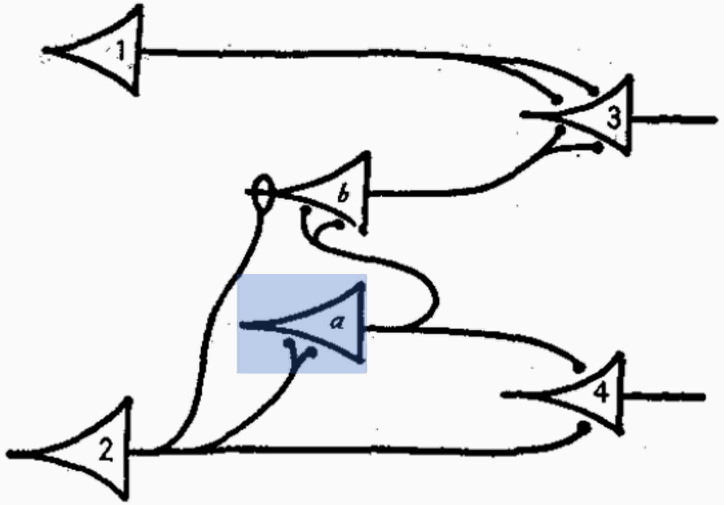
"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



$N_3(t) = N_1(t-1) \wedge \neg N_2(t-1)$



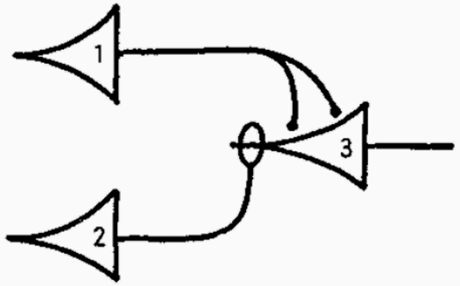
$N_3(t) = N_1(t-1) \vee \left(\neg N_2(t-2) \wedge N_2(t-3)\right)$
$N_4(t) = N_2(t-1) \wedge N_2(t-2)$

But wait there's more…

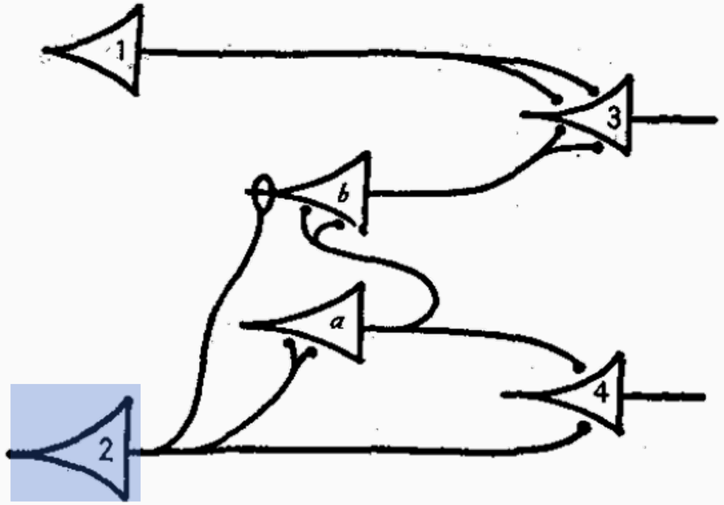
"The method […] does in fact provide a very convenient and workable procedure for constructing nervous nets to order…"

The paper shows how to construct neural networks for any Boolean function

Some examples:



$$N_3(t) = N_1(t-1) \wedge \neg N_2(t-1)$$



$$N_3(t) = N_1(t-1) \vee \left(\neg N_2(t-2) \wedge N_2(t-3)\right)$$
$$N_4(t) = N_2(t-1) \wedge N_2(t-2)$$

This was an important paper

Introduced artificial neural networks

- Time plays an important role in the design of the networks
- Describes neural networks with loops as a mechanism to model memory

Showed that a network consisting of McCulloch-Pitts neurons can compute exactly those functions as a Turing machine with a finite tape

Influenced subsequent research into automata and logic. Some examples:

- John von Neumann’s work on digital computers & theory of automata
- Stephen Kleene invented regular expressions in an attempt to describe a certain subset of McCulloch-Pitts neural networks (They used the term ‘prehensible’ to describe the sets)
- Perceptrons built on top of these ideas

Lecture outline

- Conjunctions, Disjunctions and Boolean functions as threshold networks
- The McCulloch-Pitts paper
- Knowledge-Based Artificial Neural Networks
- Augmenting neural networks with logic

Domain theories versus example-based learning

Suppose you want to teach a student to recognize members of a certain class

Domain theories versus example-based learning

Suppose you want to teach a student to recognize members of a certain class

Approach 1

Define a *domain theory* that describes:

- how to recognize critical facets of class members
- how those facets interact

Use this domain theory to teach the student to distinguish between members and nonmembers of the class

Domain theories versus example-based learning

Suppose you want to teach a student to recognize members of a certain class

Approach 1

Define a *domain theory* that describes:

- how to recognize critical facets of class members
- how those facets interact

Use this domain theory to teach the student to distinguish between members and nonmembers of the class

Approach 2

Show the student many examples of objects, one at a time

- For each example, tell the student whether it is or is not a member of the class

After seeing sufficient examples, the student can identify new examples

Domain theories versus example-based learning

Suppose you want to teach a student to recognize members of a certain class

Approach 1

Define a *domain theory* that describes:

- how to recognize critical facets of class members
- how those facets interact

Use this domain theory to teach the student to distinguish between members and nonmembers of the class

Hand-built classifiers

Approach 2

Show the student many examples of objects, one at a time

- For each example, tell the student whether it is or is not a member of the class

After seeing sufficient examples, the student can identify new examples

Empirical learning

Knowledge-Based Artificial Neural Networks (KBANN)

Towell, Geoffrey G., and Jude W. Shavlik. 1994. “Knowledge-Based Artificial Neural Networks.” *Artificial Intelligence* 70 (1–2): 119–65

A hybrid that combines domain theories with learned systems

The high level approach:

1. Translate the domain rules into a neural network
2. Train the network using backpropagation

Knowledge-Based Artificial Neural Networks (KBANN)

A hybrid that combines domain theories with learned systems

The high level approach:

1. Translate the domain rules into a neural network
2. Train the network using backpropagation

Correspondences between knowledge-bases and neural networks.

Knowledge Base		*Neural Network*
Final Conclusions	$\Longleftrightarrow$	Output Units
Supporting Facts	$\Longleftrightarrow$	Input Units
Intermediate Conclusions	$\Longleftrightarrow$	Hidden Units
Dependencies	$\Longleftrightarrow$	Weighted Connections

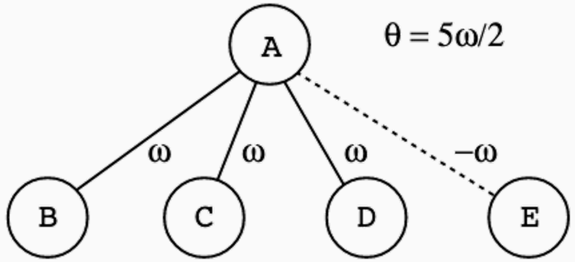
Converting knowledge bases into neural networks

At a high level, similar to what we have already seen

Works with a knowledge specified as **Horn clauses**

$B \wedge C \wedge D \wedge \neg E \rightarrow A$

```
A :- B, C, D, not(E).
```



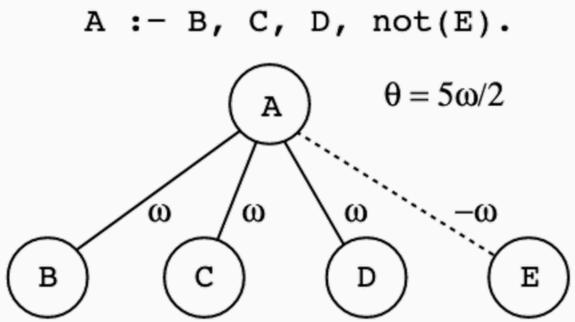
Translation of a conjunctive rule into a KBANN-net.

Converting knowledge bases into neural networks

At a high level, similar to what we have already seen

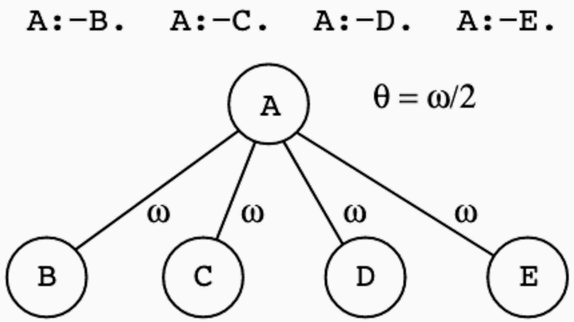
Works with a knowledge specified as **Horn clauses**

$B \wedge C \wedge D \wedge \neg E \rightarrow A$



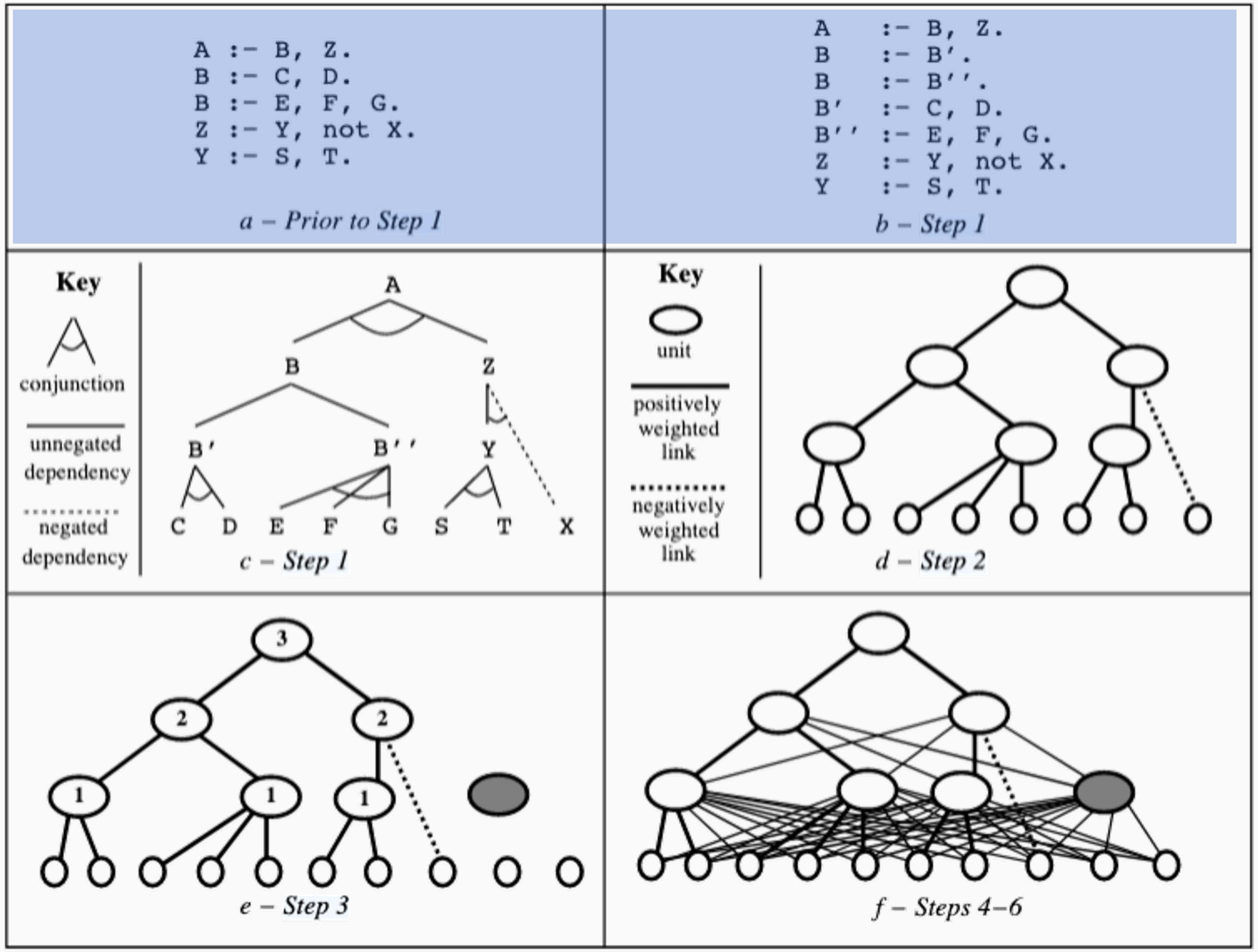
Translation of a conjunctive rule into a KBANN-net.

$B \vee C \vee D \vee E \rightarrow A$



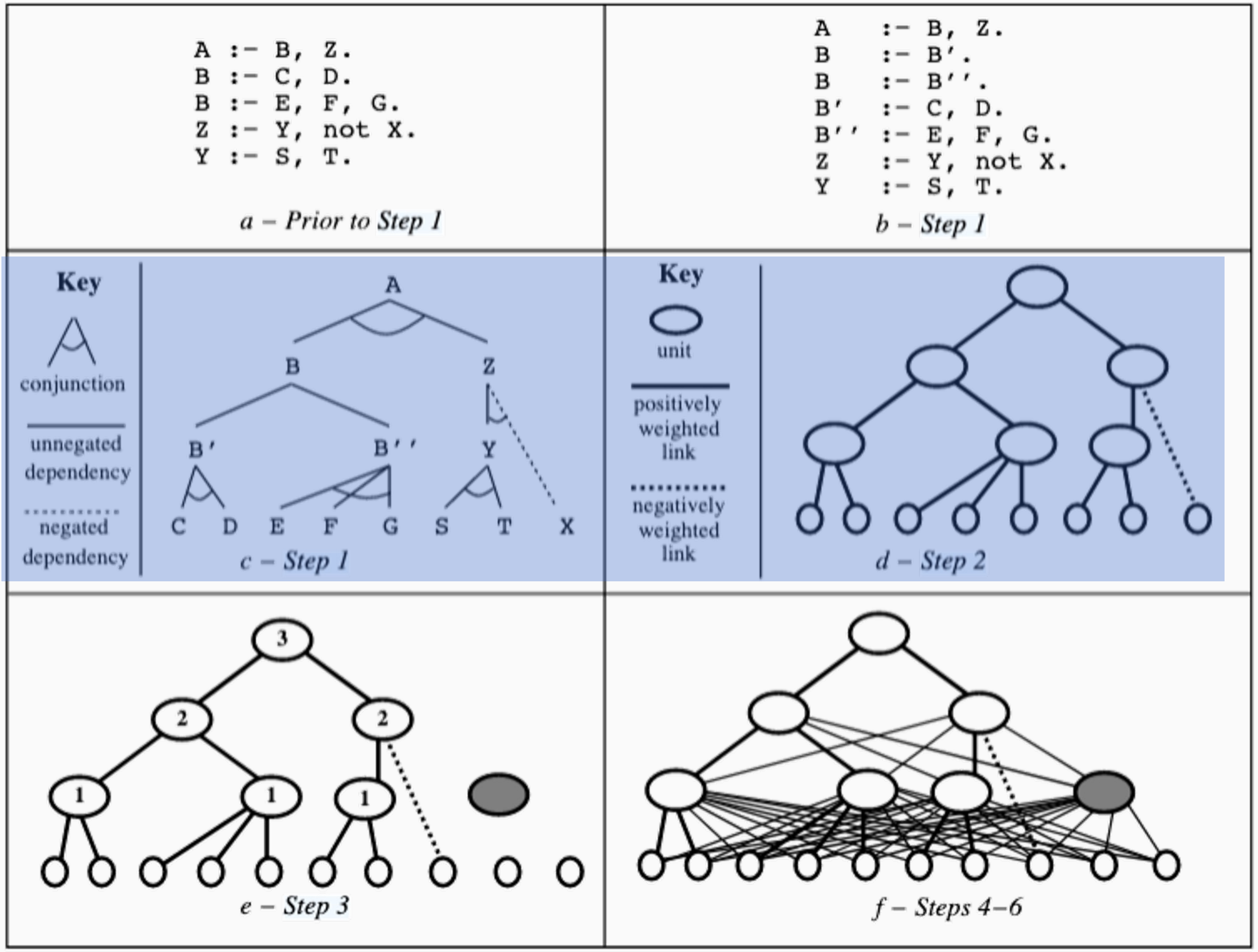
Translation of disjunctive rules into a KBANN-net.

Illustration of rules-to-network translation



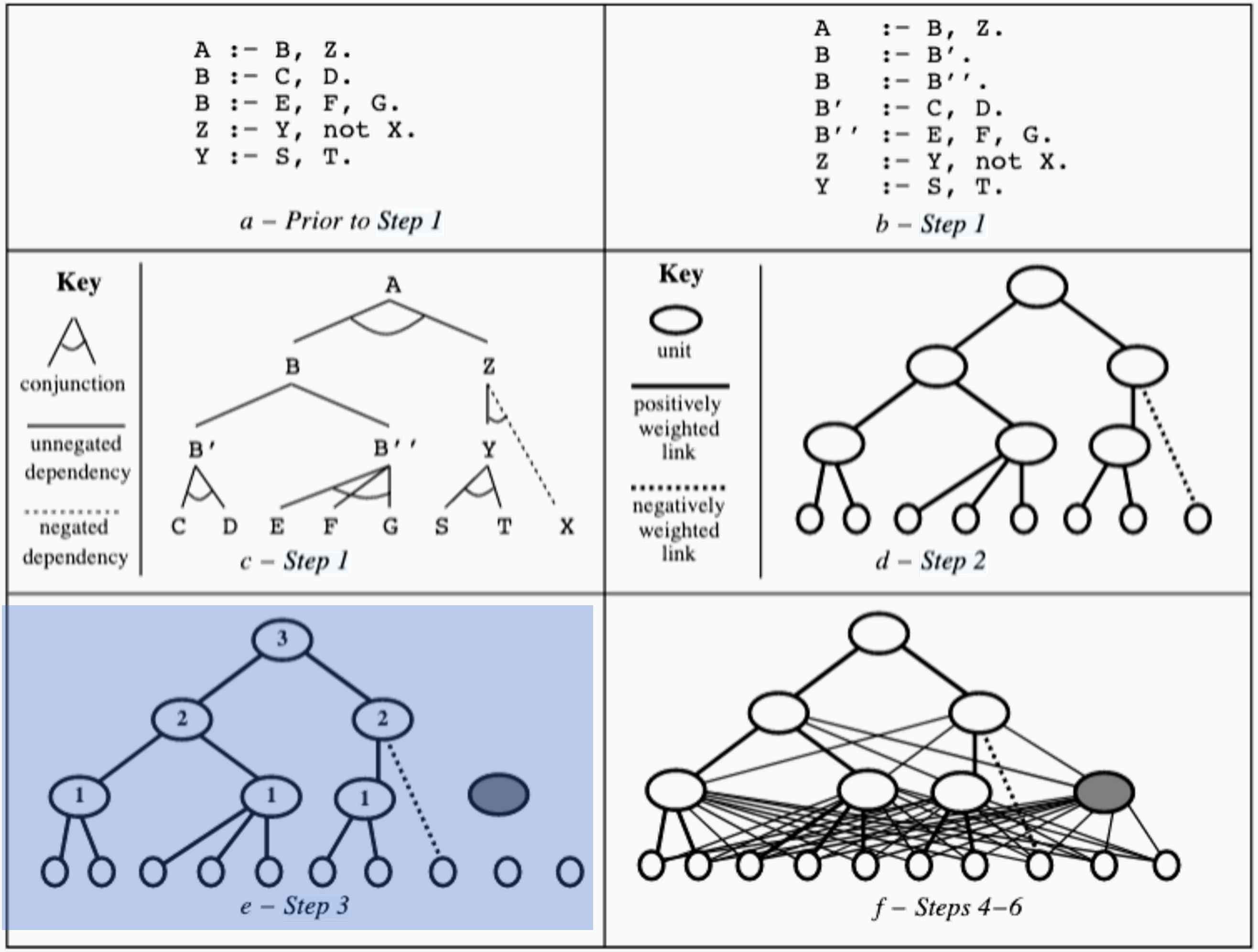
1. Normalize the knowledge base

Illustration of rules-to-network translation



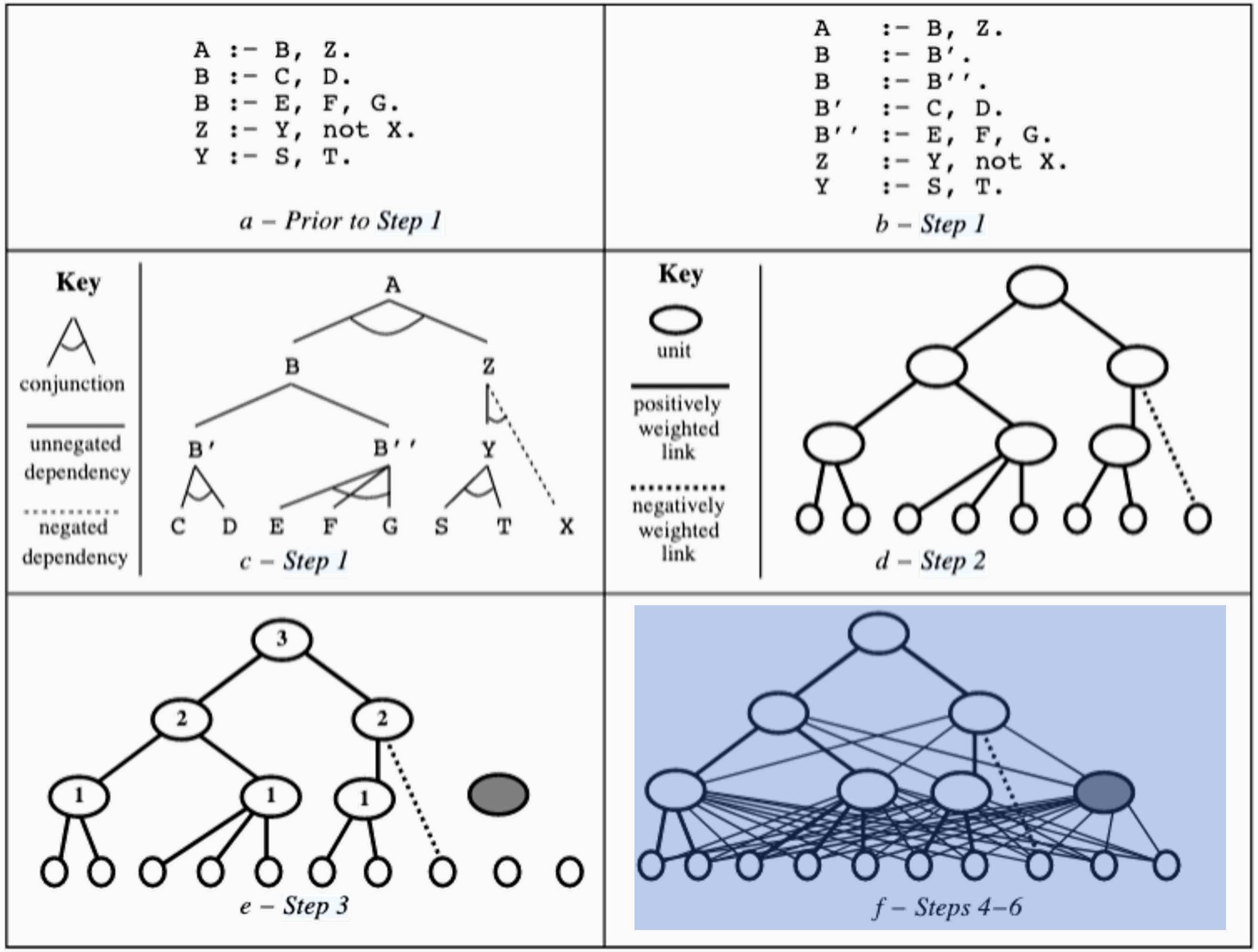
1. Normalize the knowledge base
2. Convert the KB into a network using the rules for conjunctions and disjunctions

Illustration of rules-to-network translation



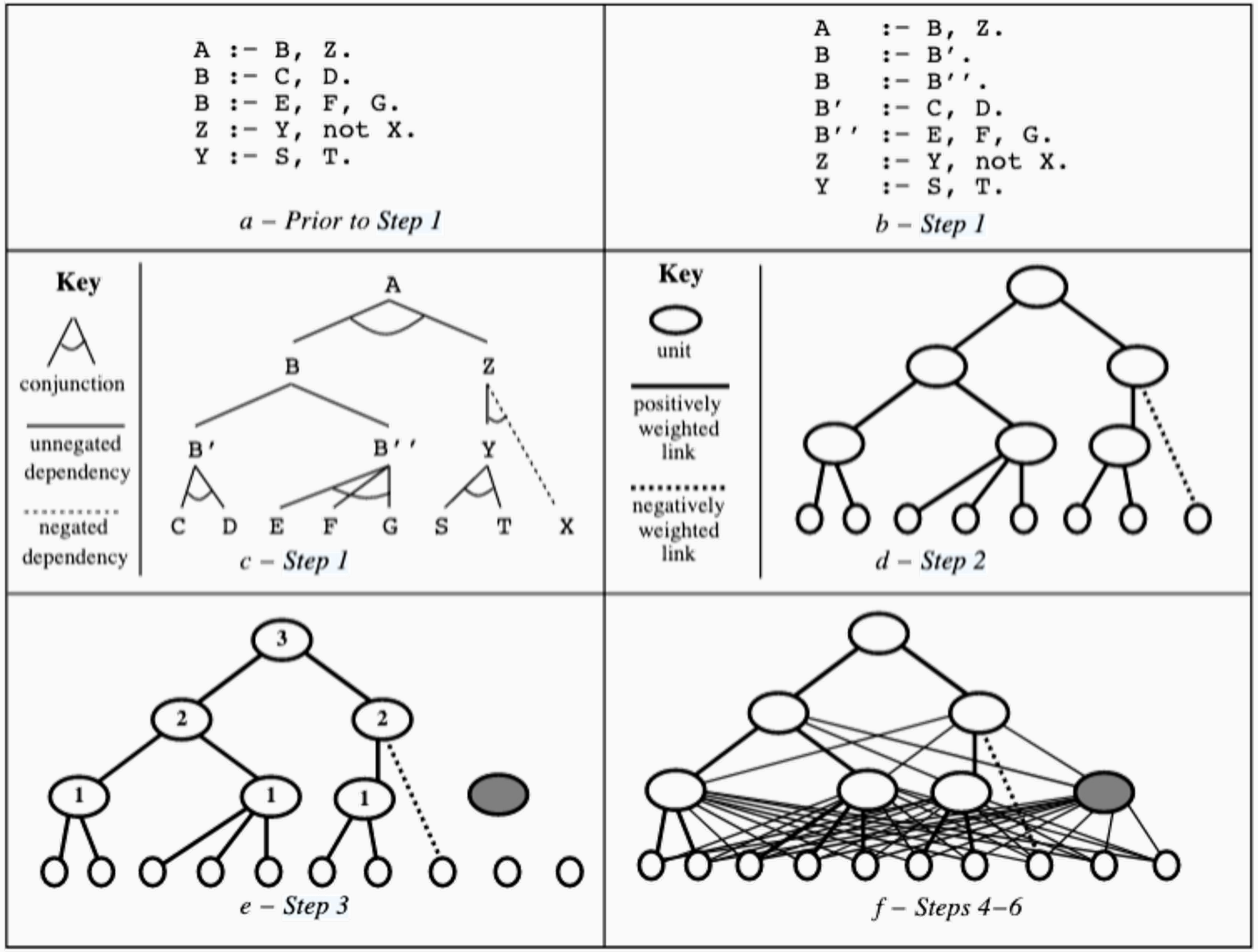
1. Normalize the knowledge base
2. Convert the KB into a network using the rules for conjunctions and disjunctions
3. Add additional hidden nodes if necessary

Illustration of rules-to-network translation



1. Normalize the knowledge base
2. Convert the KB into a network using the rules for conjunctions and disjunctions
3. Add additional hidden nodes if necessary
4. Add all other edges between pairs of layers, with small random weights

Illustration of rules-to-network translation



1. Normalize the knowledge base
2. Convert the KB into a network using the rules for conjunctions and disjunctions
3. Add additional hidden nodes if necessary
4. Add all other edges between pairs of layers, with small random weights

This final network is ready to train on data

A more real example involving a genomics application

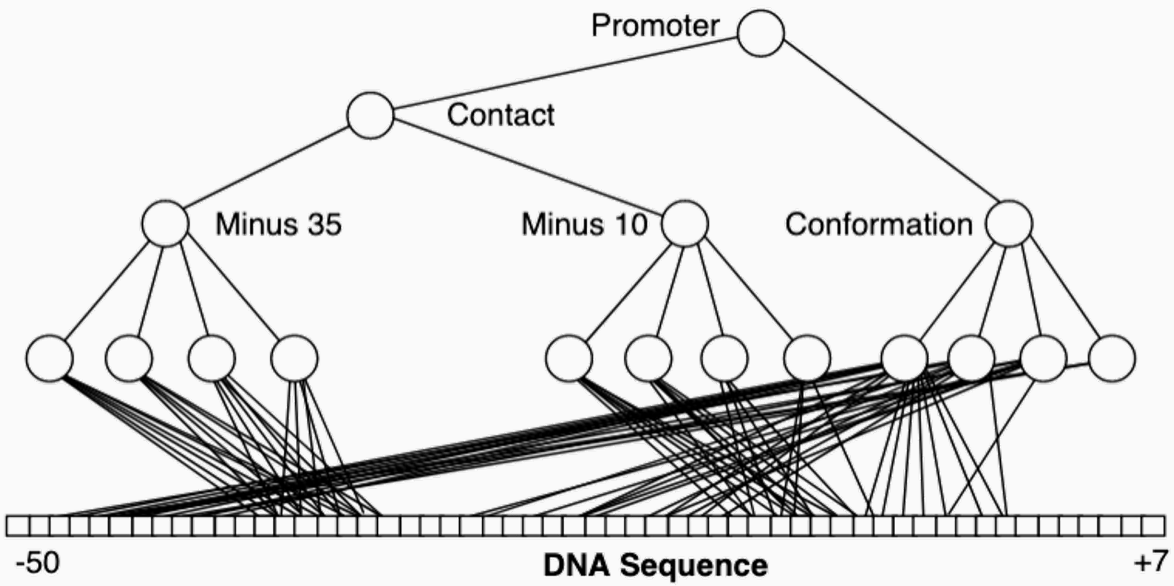
```
promoter   :- contact, conformation.
contact    :- minus-35, minus-10.

minus-35 :- @-37 'CTTGAC'.
minus-35 :- @-36  'TTGACA'.
minus-35 :- @-36  'TTG*CA'.
minus-35 :- @-36  'TTGAC'.

minus-10 :- @-13  'TA*A*T'.
minus-10 :- @-12   'TA***T'.
minus-10 :- @-14 'TATAAT'.
minus-10 :- @-13  'TATAAT'.

conformation  :-  @-45     'AA**A'.
conformation  :-  @-45     'A***A', @-28 'T***T*AA**T', @-04 'T'.
conformation  :-  @-49 'A****T',    @-27  'T****A**T*TG', @-01 'A'.
conformation  :-  @-47   'CAA*TT*AC', @-22 'G***T*C', @-08 'GCGCC*CC'
```

Gets mapped to



Key observations

When there’s a limited amount of data, KBANN outperforms a knowledge-agnostic network

The approach constructs the structure of the network and assigns initial weights for some edges. Both factors are important

Limitation: Cannot handle rules that have cycles in them

The eventual learned network may overrule the initial weights that come from the rules. Yet the rules help empirically

Lecture outline

- Conjunctions, Disjunctions and Boolean functions as threshold networks
- The McCulloch-Pitts paper
- Knowledge-Based Artificial Neural Networks
- Augmenting neural networks with logic

Is it realistic to build entire networks using logic?

What are some disadvantages?

Is it realistic to build entire networks using logic?

What are some disadvantages?

We may have an existing neural network architecture for a task

– Don't want to get rid of something that works

The logical rules may be incomplete

– For complex phenomena, maybe there is no complete symbolic description

The logical rules may be incorrect

– Maybe they were derived using a theoretical framework that is not correct

The rules may be only partially correct

– Maybe they are to be treated as soft constraints that data should be allowed to override

Can we integrate rules into an existing network?

Not always possible

We will assume that some nodes in the network are *named neurons*
(i.e. some nodes have externally defined semantics)

We will write rules about these nodes

Goal: Integrate rules into a network that can be trained end-to-end

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates

Li, Tao, and Vivek Srikumar. "Augmenting Neural Networks with First-order Logic." In *ACL 2019*

The intuition

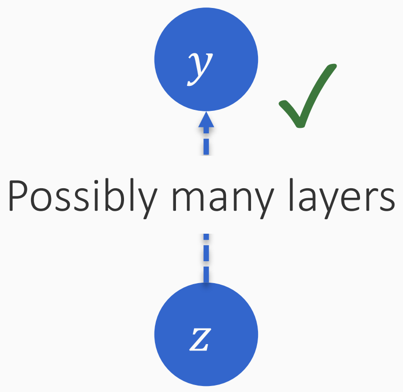
Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates

Important assumption: The node y is downstream of the node z in the network

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates

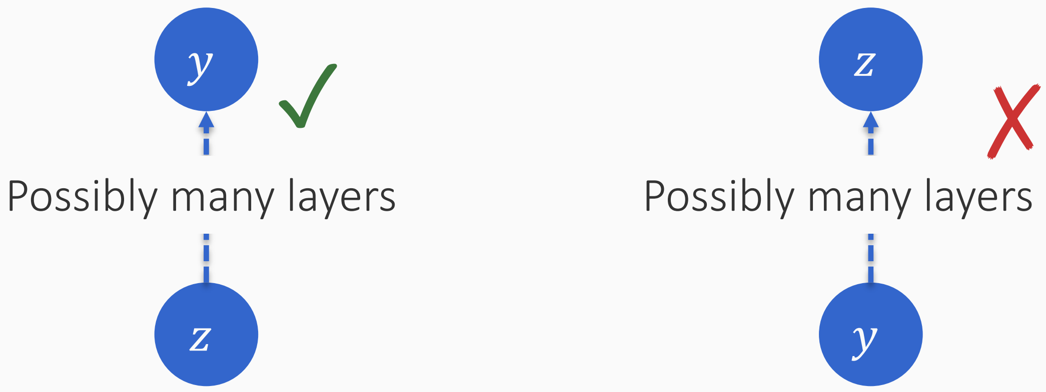
Important assumption: The node y is downstream of the node z in the network



The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates

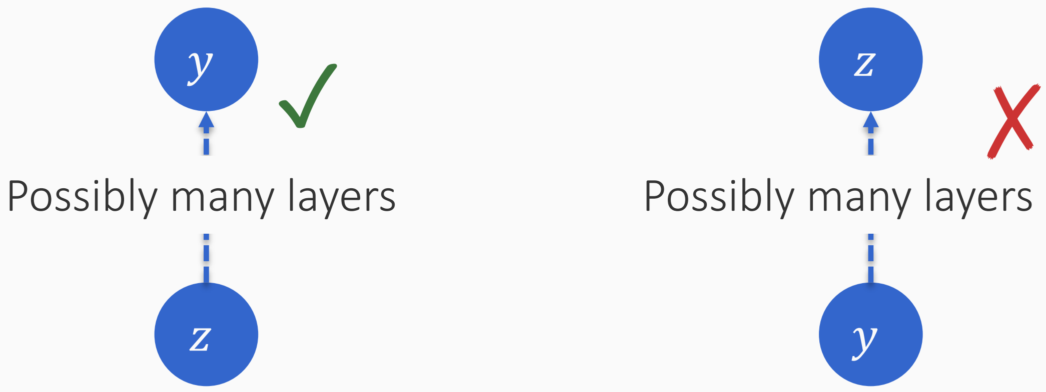
Important assumption: The node y is downstream of the node z in the network



The intuition

Suppose we have a rule $Z \to Y$ and z and y are nodes in a neural network that correspond to these predicates

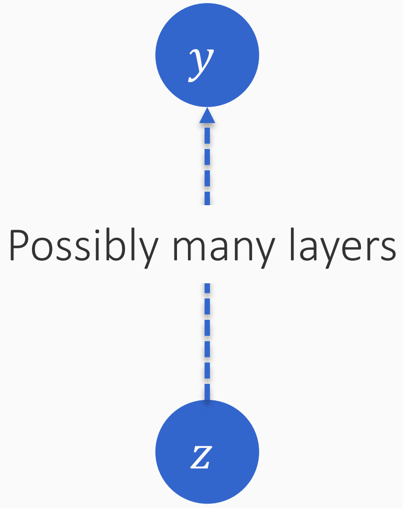
Important assumption: The node y is downstream of the node z in the network



Such rules are called ***acyclic***

The intuition

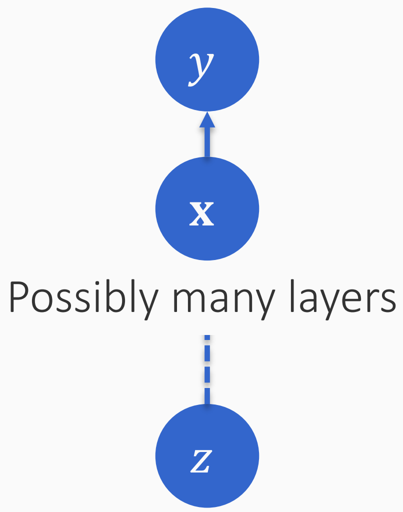
Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The node z need not be directly connected to the node y

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



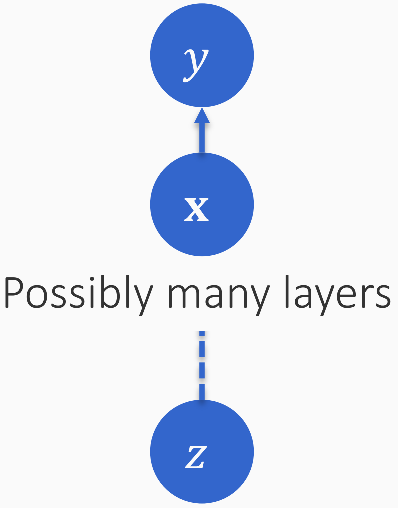
The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

The node z need not be directly connected to the node y

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

Can we change something in the architecture that enforces this? Ideas?

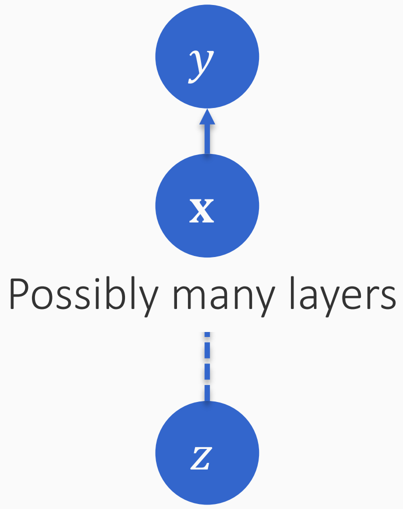
If z is 1, then y should be 1

The node z need not be directly connected to the node y

If z is 0, then the rule doesn't say anything about y

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T \mathbf{x})$$

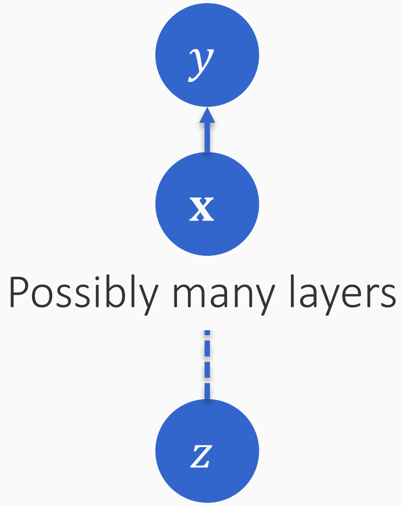
The node z need not be directly connected to the node y

If z is 1, then y should be 1 → The logit for y should be infinite

If z is 0, then the rule doesn't say anything about y

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

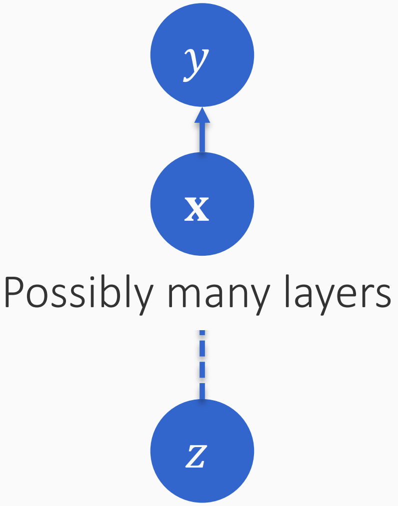
The node z need not be directly connected to the node y

If z is 1, then y should be 1 → The logit for y should be infinite

If z is 0, then the rule doesn't say anything about y → The logit for y should be whatever the rest of the network says

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The node z need not be directly connected to the node y

The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

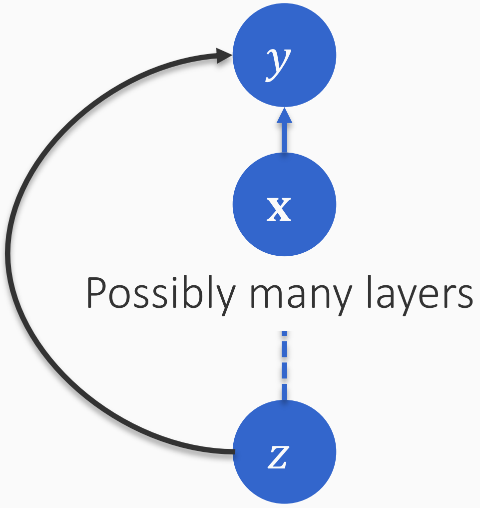
Can we change something in the architecture that enforces this? Ideas?

If z is 1, then y should be 1 → The logit for y should be infinite

If z is 0, then the rule doesn't say anything about y → The logit for y should be whatever the rest of the network says

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The node z need not be directly connected to the node y

The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

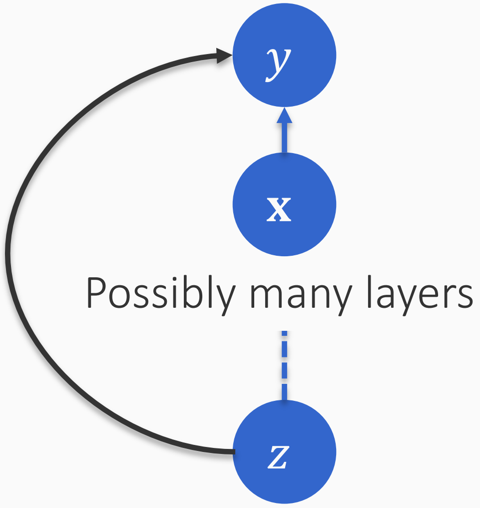
→

The immediate input layer to y is a layer $\mathbf{x}$ and the node z

$$y = \sigma(\mathbf{w}^T\mathbf{x} + \rho d(z))$$

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The node z need not be directly connected to the node y

The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T \mathbf{x})$$

→

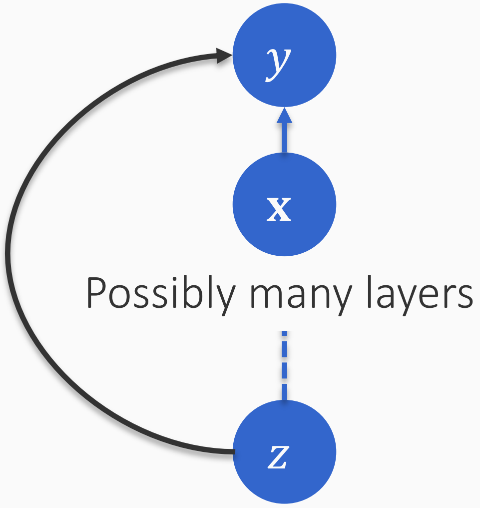
The immediate input layer to y is a layer $\mathbf{x}$ and the node z

$$y = \sigma(\mathbf{w}^T \mathbf{x} + \rho d(z))$$

Takes value 1 if the LHS of the rule is true

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The node z need not be directly connected to the node y

The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

→

The immediate input layer to y is a layer $\mathbf{x}$ and the node z

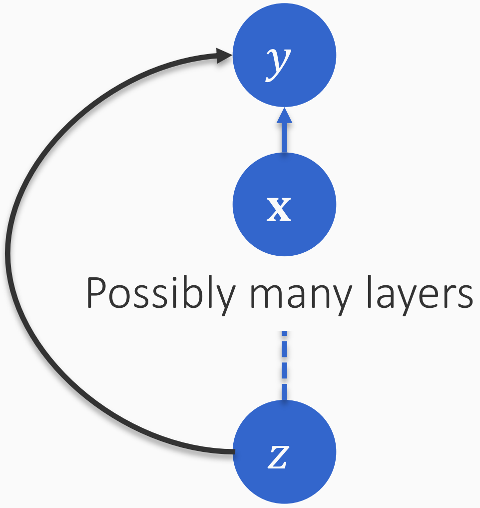
$$y = \sigma(\mathbf{w}^T\mathbf{x} + \rho d(z))$$

Takes value 1 if the LHS of the rule is true

$$d(z) = \begin{cases} 1, & \text{Z holds} \\ 0, & \text{else} \end{cases}$$

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The node z need not be directly connected to the node y

The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

→

The immediate input layer to y is a layer $\mathbf{x}$ and the node z

$$y = \sigma(\mathbf{w}^T\mathbf{x} + \rho d(z))$$

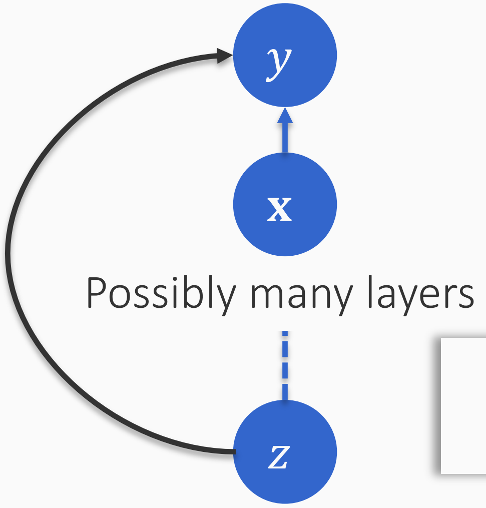
A large positive number to make the logit infinite of the LHS is true

Takes value 1 if the LHS of the rule is true

$$d(z) = \begin{cases} 1, & \text{Z holds} \\ 0, & \text{else} \end{cases}$$

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The node z need not be directly connected to the node y

The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

Why is this problematic?

→

The immediate input layer to y is a layer $\mathbf{x}$ and the node z

$$y = \sigma(\mathbf{w}^T\mathbf{x} + \rho d(z))$$

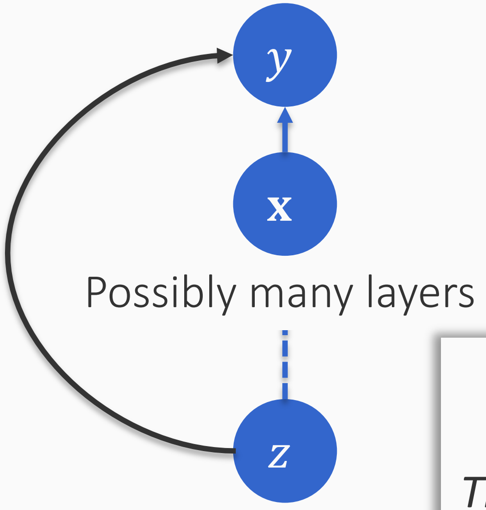
ρ: A large positive number to make the logit infinite of the LHS is true

$d(z)$: Takes value 1 if the LHS of the rule is true

$$d(z) = \begin{cases} 1, & \text{Z holds} \\ 0, & \text{else} \end{cases}$$

The intuition

Suppose we have a rule $Z \rightarrow Y$ and z and y are nodes in a neural network that correspond to these predicates



The node z need no
directly connected t
the node y

The immediate input layer to y is a layer $\mathbf{x}$

$$y = \sigma(\mathbf{w}^T\mathbf{x})$$

→

The immediate input layer to y is a layer $\mathbf{x}$ and the node z

$$y = \sigma(\mathbf{w}^T\mathbf{x} + \rho d(z))$$

ρ: A large positive number to make the logit infinite of the LHS is true

$d(z)$: Takes value 1 if the LHS of the rule is true

$$d(z) = \begin{cases} 1, & \text{Z holds} \\ 0, & \text{else} \end{cases}$$

Why is this problematic?

The function d is not differentiable for more complex LHS

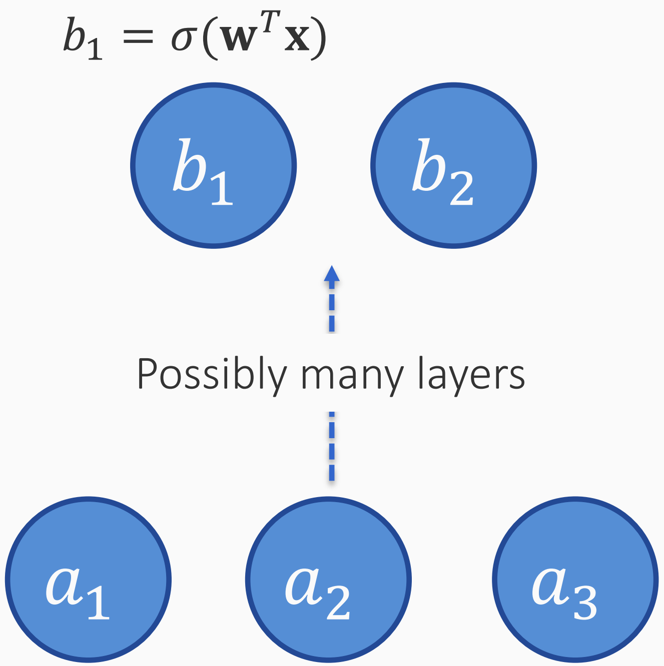
Prohibits end-to-end training

Using differentiable rules

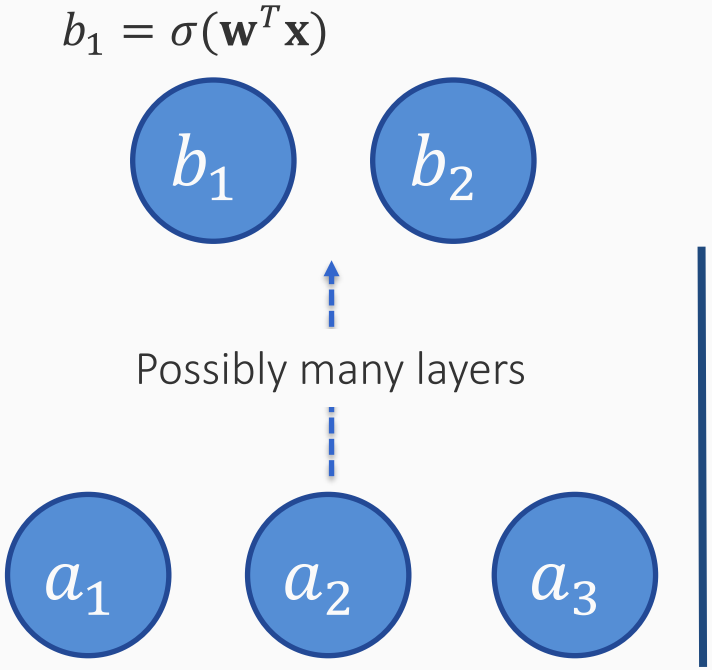
The solution: Relax the LHS with Lukasiewicz logic

Antecedent	Distance $d(\mathbf{z})$
$\bigwedge_i Z_i$	$\max(0, \sum_i z_i - \lvert Z \rvert + 1)$
$\bigvee_i Z_i$	$\min(1, \sum_i z_i)$
$\neg \bigvee_i Z_i$	$\max(0, 1 - \sum_i z_i)$
$\neg \bigwedge_i Z_i$	$\min(1, N - \sum_i z_i)$

Augmenting models: An example

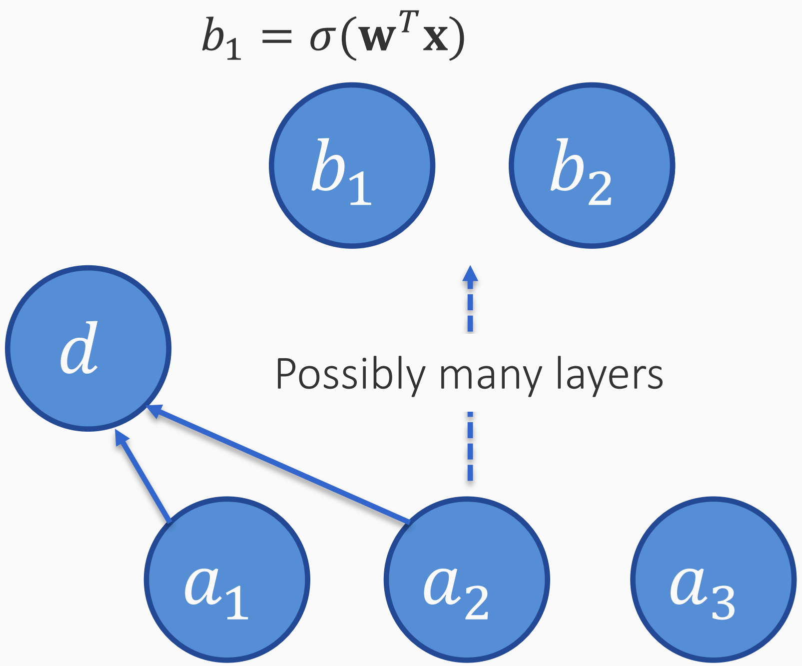


Augmenting models: An example



$$A_1 \land A_2 \rightarrow B_1$$

Augmenting models: An example

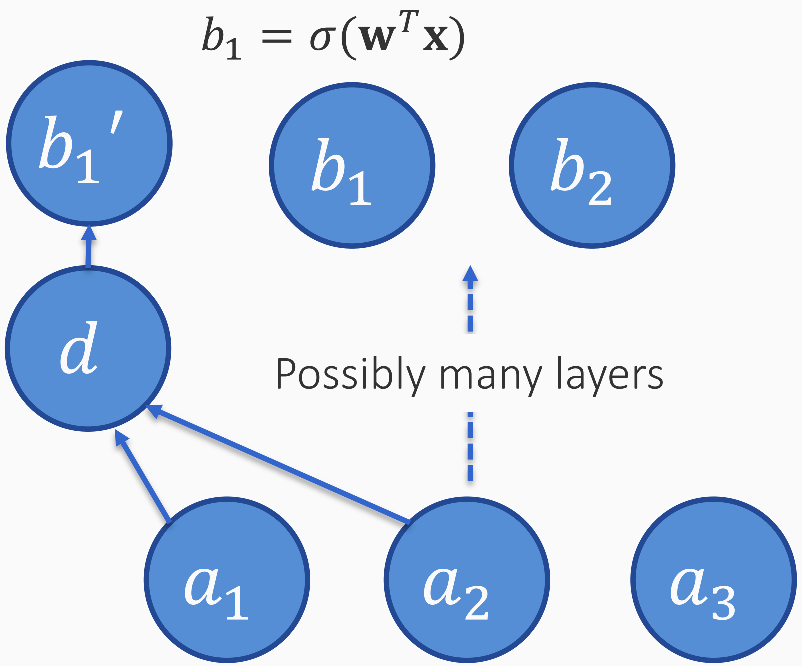


$$A_1 \land A_2 \rightarrow B_1$$

Step 1: LHS in Łukasiewicz logic

$$d(a_1, a_2) = \max(0, a_1 + a_2 - 1)$$

Augmenting models: An example



$$A_1 \wedge A_2 \rightarrow B_1$$

Step 1: LHS in Łukasiewicz logic

$$d(a_1, a_2) = \max(0, a_1 + a_2 - 1)$$

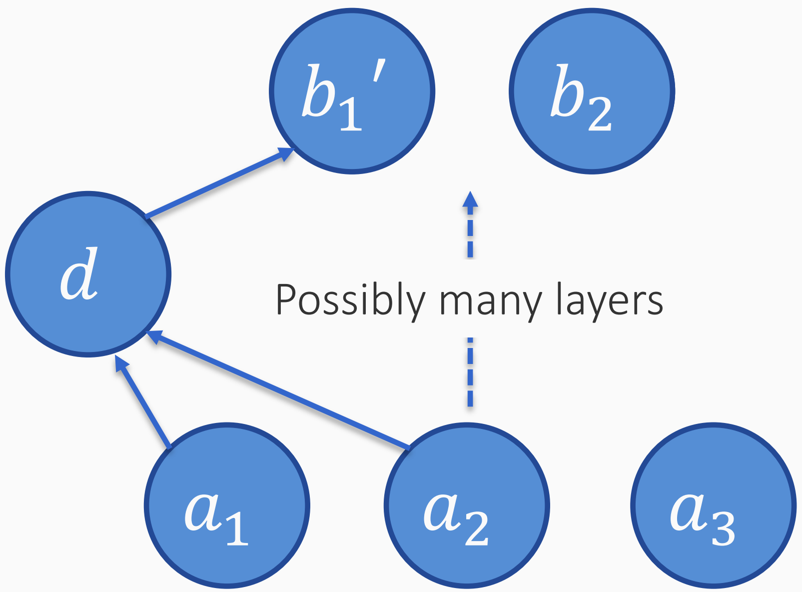
Step 2: Define constrained node:

$$b_1' = \sigma(\mathbf{w}^T\mathbf{x} + \rho\, d(a_1, a_2))$$

Augmenting models: An example

$b_1' = \sigma(\mathbf{w}^T\mathbf{x} + \rho\, d(a_1, a_2))$

~~$b_1 = \sigma(\mathbf{w}^T\mathbf{x})$~~



$$A_1 \wedge A_2 \rightarrow B_1$$

Step 1: LHS in Łukasiewicz logic

$$d(a_1, a_2) = \max(0, a_1 + a_2 - 1)$$

Step 2: Define constrained node:

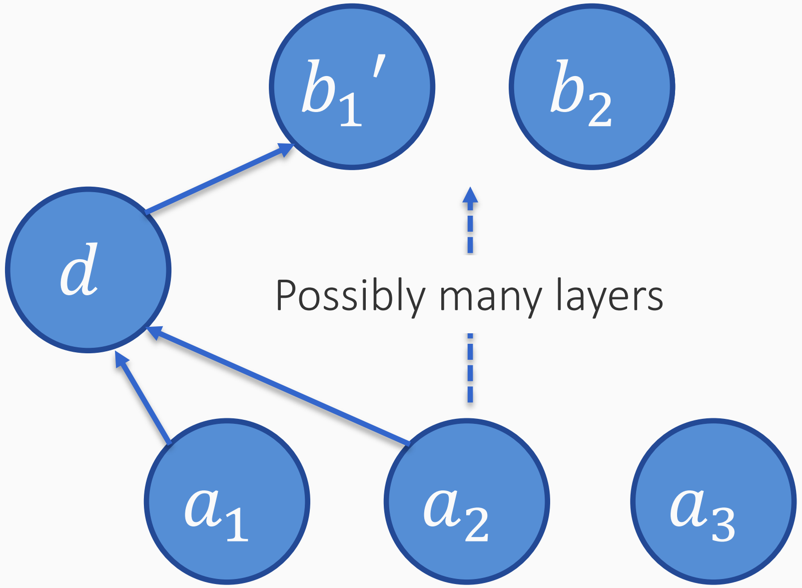
$$b_1' = \sigma(\mathbf{w}^T\mathbf{x} + \rho\, d(a_1, a_2))$$

Step 3: Replace original b_1 with b_1'

Augmenting models: An example

$b_1{}' = \sigma(\mathbf{w}^T\mathbf{x} + \rho\, d(a_1, a_2))$

~~$b_1 = \sigma(\mathbf{w}^T\mathbf{x})$~~



$$A_1 \wedge A_2 \rightarrow B_1$$

Step 1: LHS in Łukasiewicz logic

$$d(a_1, a_2) = \max(0, a_1 + a_2 - 1)$$

Step 2: Define constrained node:

$$b_1{}' = \sigma(\mathbf{w}^T\mathbf{x} + \rho\, d(a_1, a_2))$$

Step 3: Replace original b_1 with $b_1{}'$

No additional trainable parameters introduced

Hyperparameter ρ controls how strongly the constraint is enforced

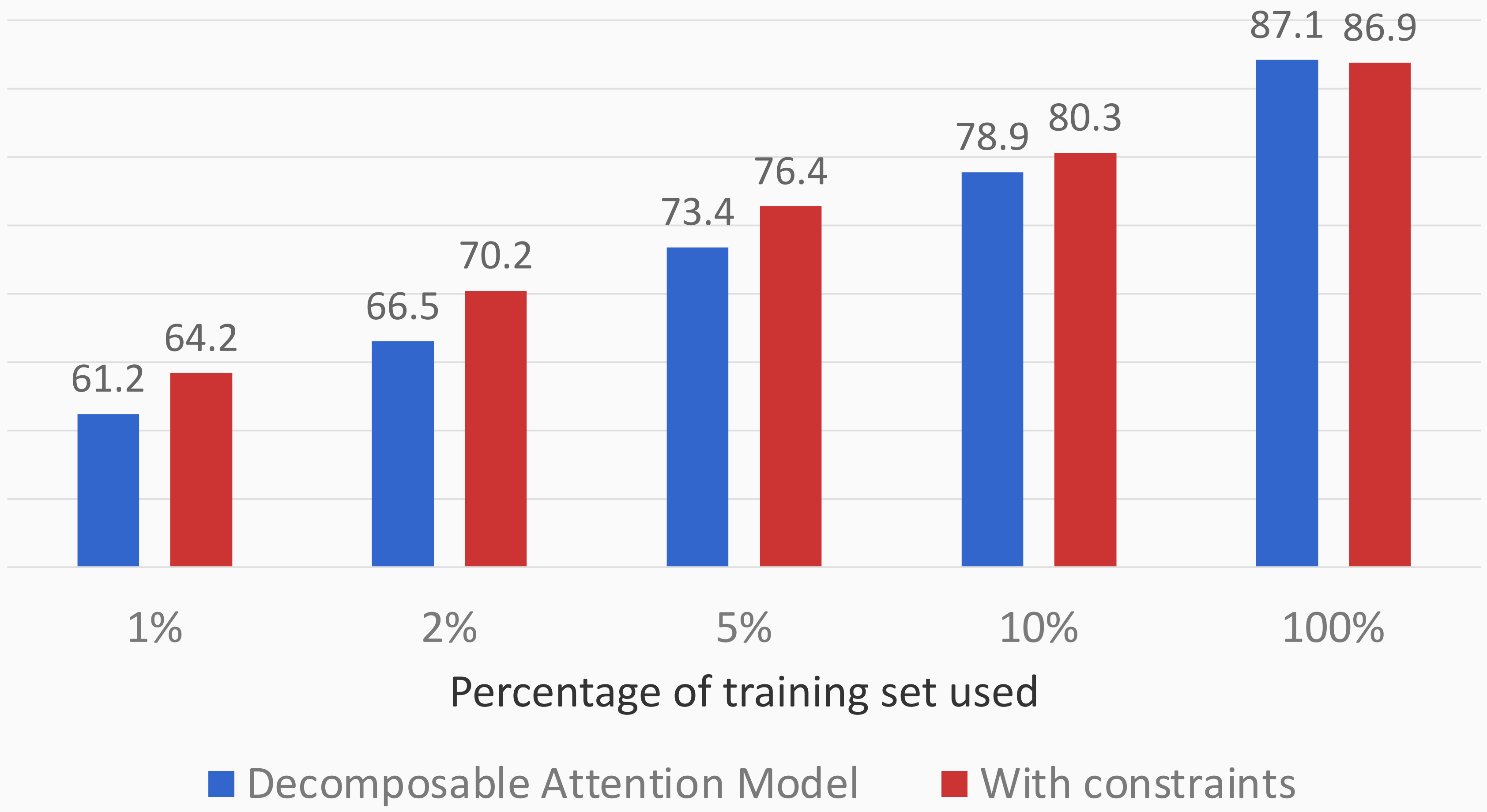
Natural Language Inference

SNLI dataset, decomposable attention model [Parikh et al 2016]

Two constraints (formalized in first order logic):

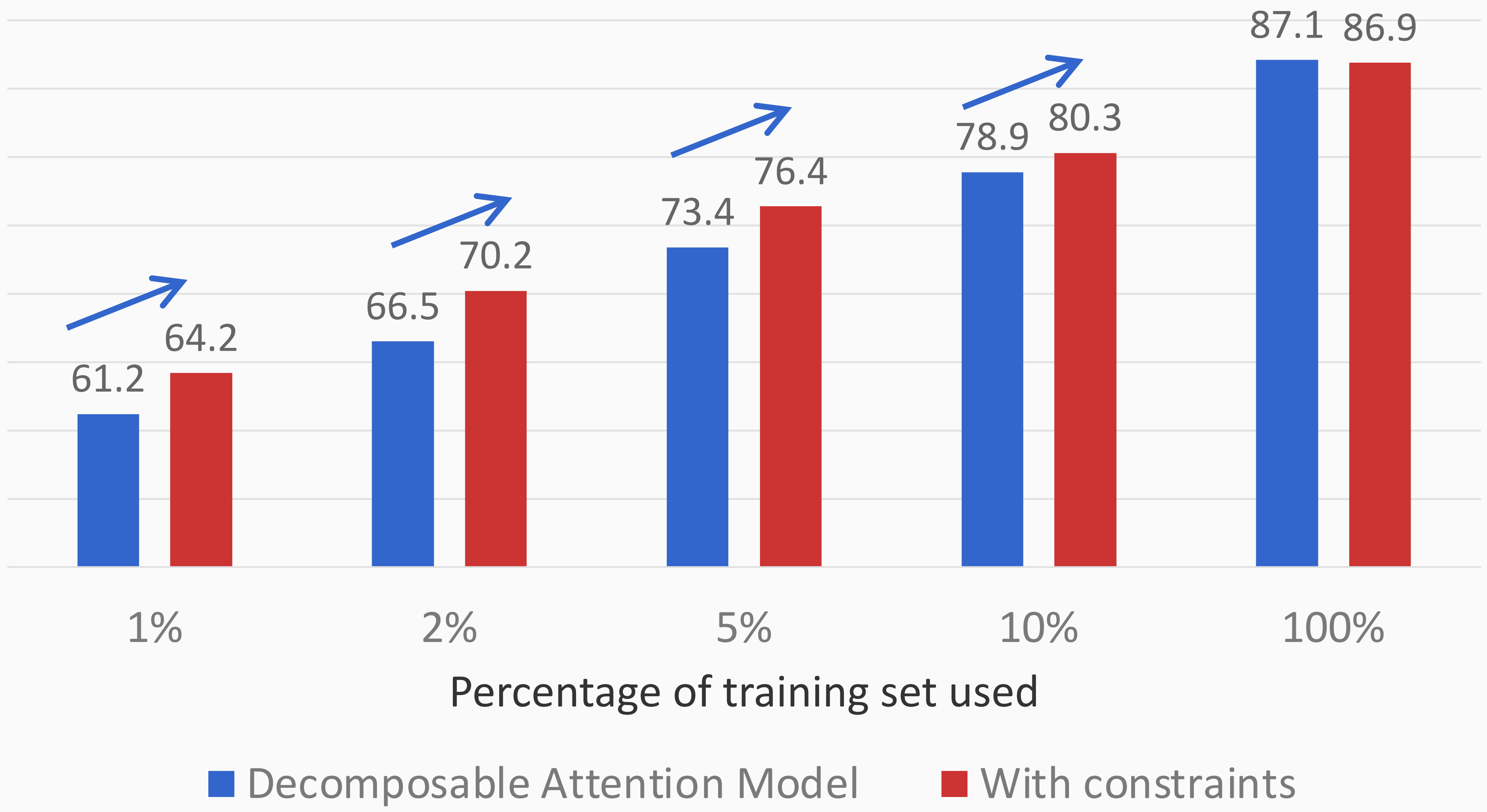
1. If two words are related, they should be aligned
2. If no content word in the hypothesis is aligned, then the label cannot be Entail

Results: Natural Language Inference



Results: Natural Language Inference

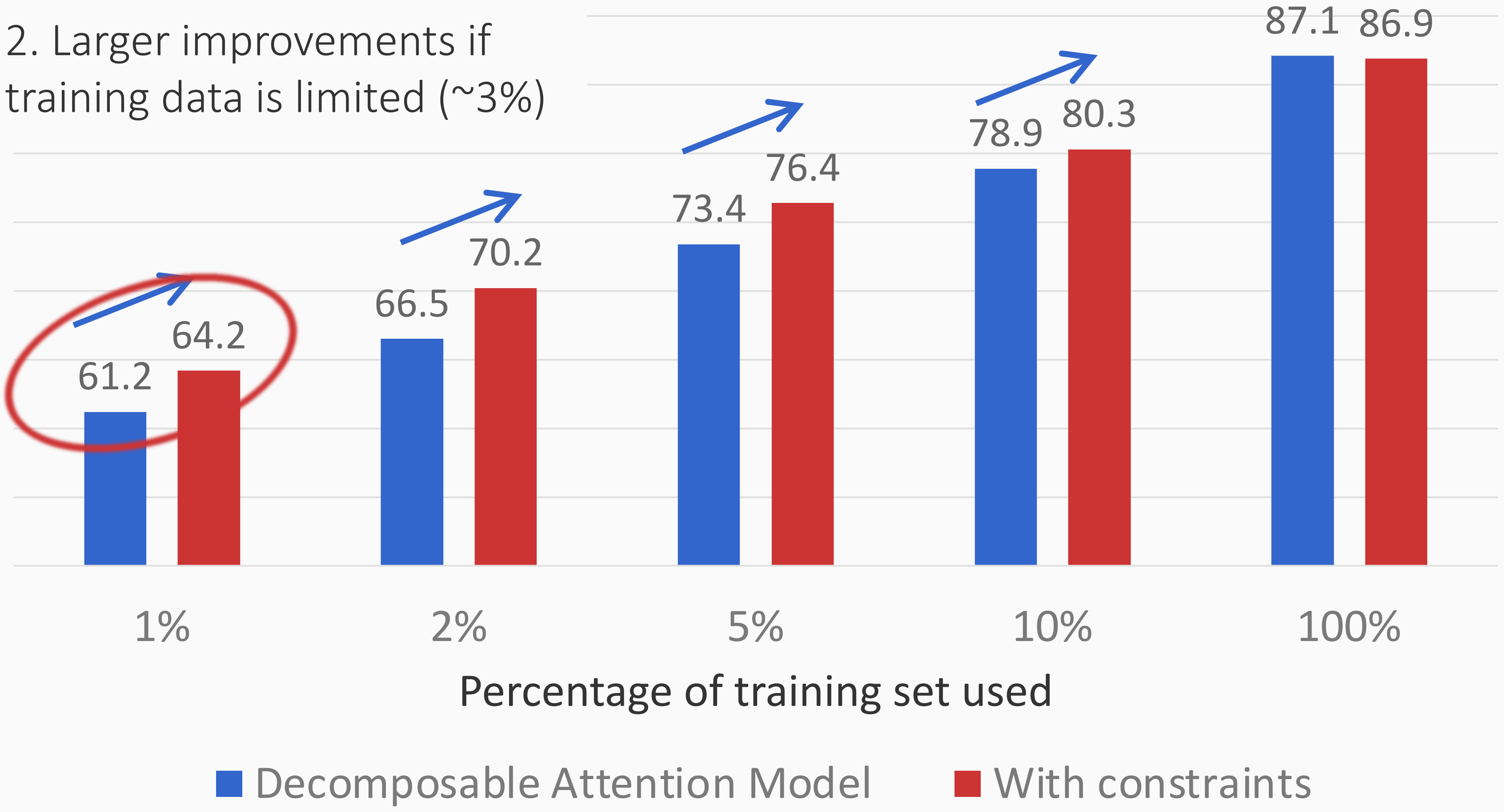
1. Constraints help



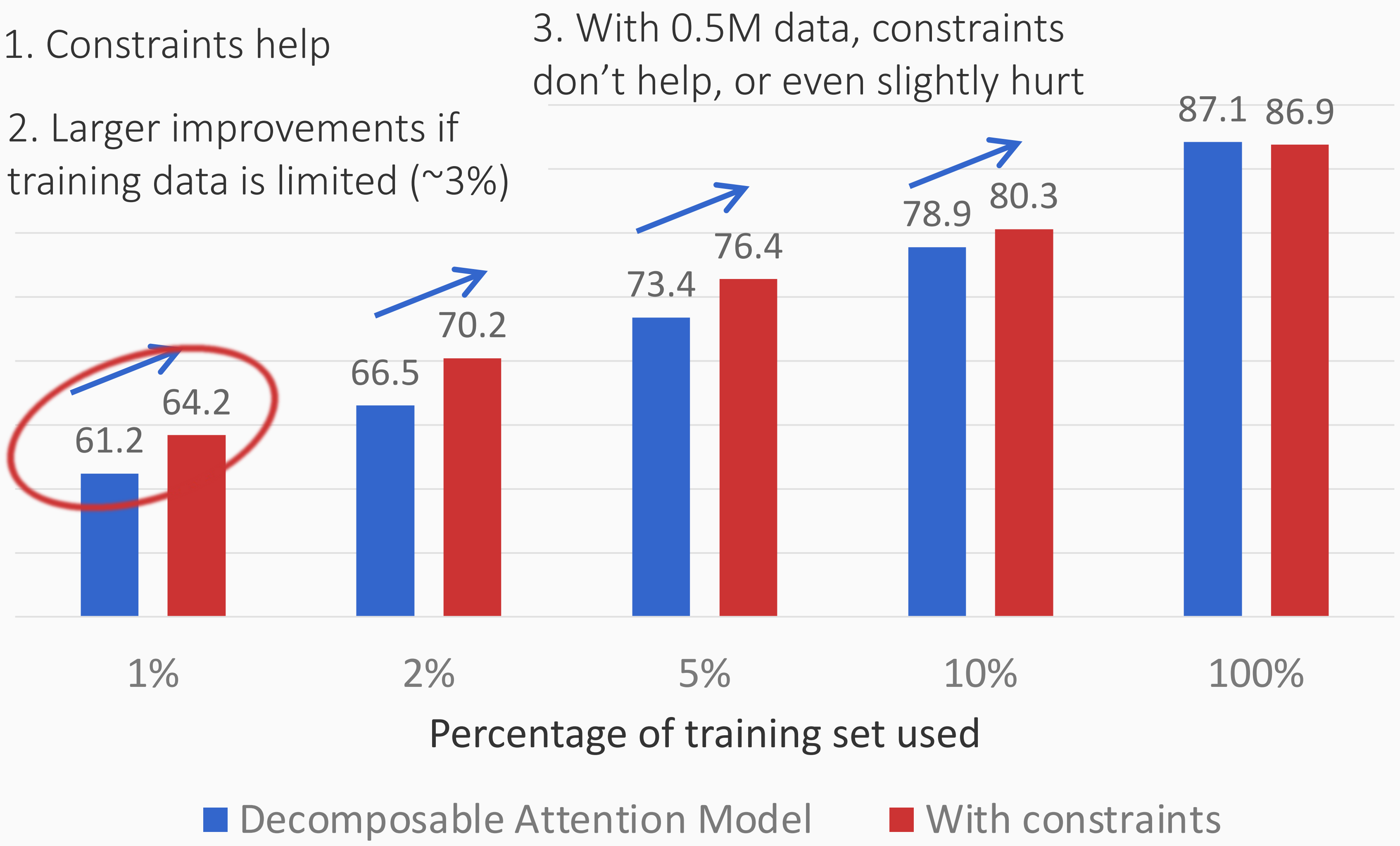
Results: Natural Language Inference

1. Constraints help

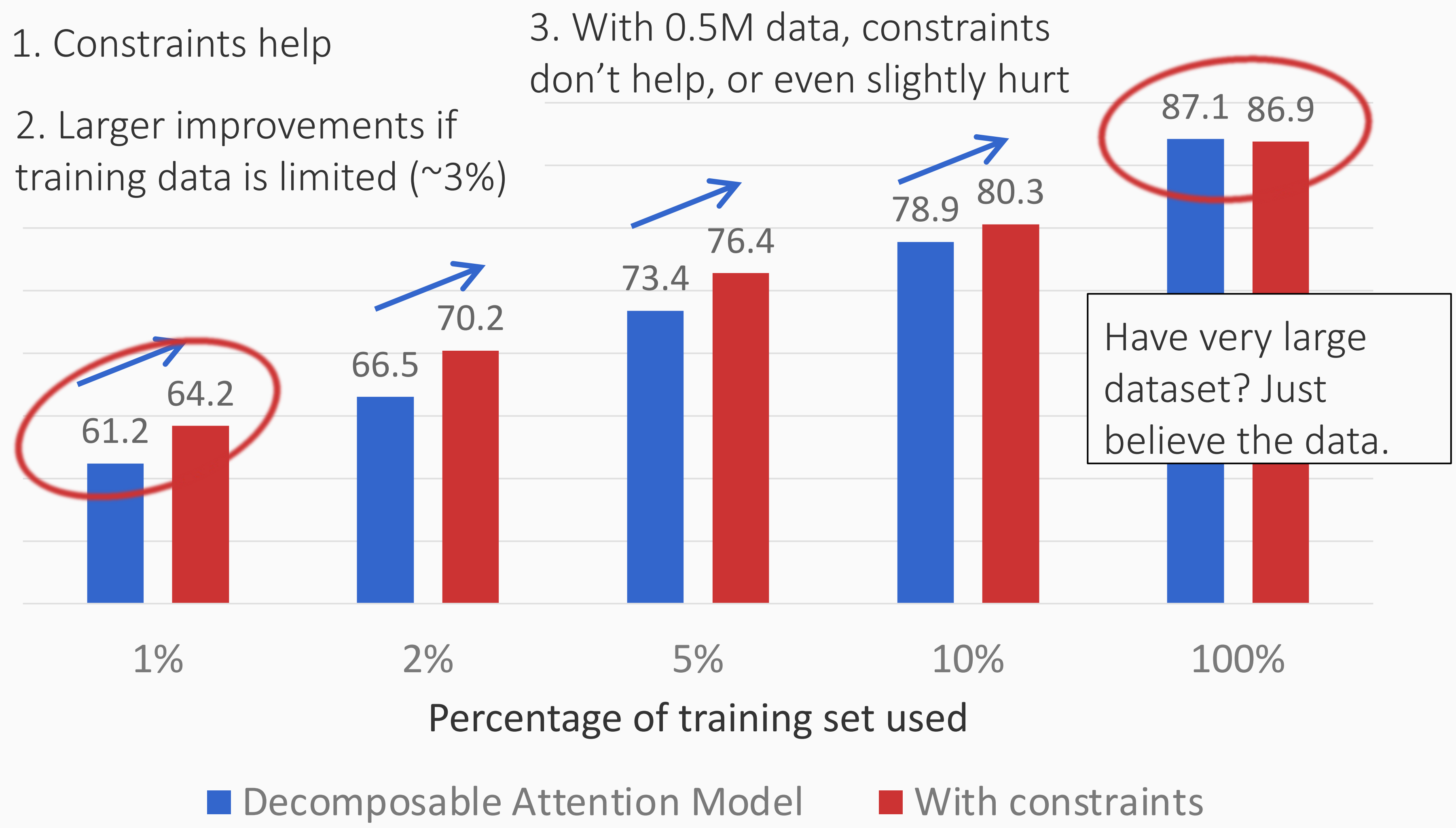
2. Larger improvements if training data is limited (~3%)



Results: Natural Language Inference



Results: Natural Language Inference



Difficulties with this approach

What are some shortcomings of the idea of constructing or editing neural network architectures with knowledge?

Difficulties with this approach

What are some shortcomings of the idea of constructing or editing neural network architectures with knowledge?

1. Acyclicity can be a strong constraint

2. Assumes that we have named neurons. Does not apply to modern transformer networks

Logic as architectures: Summary

- One of the oldest ideas in this area
  - Goes back to the original work of McCulloch & Pitts
- Key intuition: Generate or augment a neural network using rules
- Experiments show gains especially in low data regimes
- Design difficulties with modern large models