Strongly Non-U-Shaped Learning Results by General Techniques

John Case[1] **Timo Kötzing**[2]

[1] **Computer and Information Science, University of Delaware**
[2] **Max Planck Institute for Informatics**

June 28, 2010

Examples for Language Learning

We want to learn correct programs or programmable descriptions for given languages, such as:

$16, 12, 18, 2, 4, 0, 16, \ldots$	"even numbers"
$1, 16, 256, 16, 4, \ldots$	"powers of 2"
$0, 0, 0, 0, 0, \ldots$	"singleton 0"

Examples for Language Learning

We want to learn correct programs or programmable descriptions for given languages, such as:

$16, 12, 18, 2, 4, 0, 16, \ldots$	"even numbers"
$1, 16, 256, 16, 4, \ldots$	"powers of 2"
$0, 0, 0, 0, 0, \ldots$	"singleton 0"

Examples for Language Learning

We want to learn correct programs or programmable descriptions for given languages, such as:

16, 12, 18, 2, 4, 0, 16, . . .	"even numbers"
1, 16, 256, 16, 4, . . .	"powers of 2"
0, 0, 0, 0, 0, . . .	"singleton 0"

Examples for Language Learning

We want to learn correct programs or programmable descriptions for given languages, such as:

$16, 12, 18, 2, 4, 0, 16, \ldots$	"even numbers"
$1, 16, 256, 16, 4, \ldots$	"powers of 2"
$0, 0, 0, 0, 0, \ldots$	"singleton 0"

Examples for Language Learning

We want to learn correct programs or programmable descriptions for given languages, such as:

$16, 12, 18, 2, 4, 0, 16, \ldots$	"even numbers"
$1, 16, 256, 16, 4, \ldots$	"powers of 2"
$0, 0, 0, 0, 0, \ldots$	"singleton 0"

Examples for Language Learning

We want to learn correct programs or programmable descriptions for given languages, such as:

$16, 12, 18, 2, 4, 0, 16, \ldots$	"even numbers"
$1, 16, 256, 16, 4, \ldots$	"powers of 2"
$0, 0, 0, 0, 0, \ldots$	"singleton 0"

Examples for Language Learning

We want to learn correct programs or programmable descriptions for given languages, such as:

$16, 12, 18, 2, 4, 0, 16, \ldots$	“even numbers”
$1, 16, 256, 16, 4, \ldots$	“powers of 2”
$0, 0, 0, 0, 0, \ldots$	“singleton 0”

Examples for Language Learning

We want to learn correct programs or programmable descriptions for given languages, such as:

$16, 12, 18, 2, 4, 0, 16, \ldots$	“even numbers”
$1, 16, 256, 16, 4, \ldots$	“powers of 2”
$0, 0, 0, 0, 0, \ldots$	“singleton 0”

Language Learning from Positive Data

- Let $\mathbb{N} = \{0, 1, 2, \ldots\}$, the set of all natural numbers.
- A language is a set $L \subseteq \mathbb{N}$.
- A presentation for L is essentially an (infinite) listing T of all and only the elements of L. Such a T is called a text for L.
- We numerically name programs or grammars in some standard general hypothesis space, where each $e \in \mathbb{N}$ generates some language.

Language Learning from Positive Data

- Let $\mathbb{N} = \{0, 1, 2, \ldots\}$, the set of all natural numbers.
- A language is a set $L \subseteq \mathbb{N}$.
- A presentation for L is essentially an (infinite) listing T of all and only the elements of L. Such a T is called a text for L.
- We numerically name programs or grammars in some standard general hypothesis space, where each $e \in \mathbb{N}$ generates some language.

Language Learning from Positive Data

- Let $\mathbb{N} = \{0, 1, 2, \ldots\}$, the set of all natural numbers.
- A language is a set $L \subseteq \mathbb{N}$.
- A presentation for L is essentially an (infinite) listing T of all and only the elements of L. Such a T is called a text for L.
- We numerically name programs or grammars in some standard general hypothesis space, where each $e \in \mathbb{N}$ generates some language.

Language Learning from Positive Data

- Let $\mathbb{N} = \{0, 1, 2, \ldots\}$, the set of all natural numbers.
- A language is a set $L \subseteq \mathbb{N}$.
- A presentation for L is essentially an (infinite) listing T of all and only the elements of L. Such a T is called a text for L.
- We numerically name programs or grammars in some standard general hypothesis space, where each $e \in \mathbb{N}$ generates some language.

Language Learning from Positive Data

- Let $\mathbb{N} = \{0, 1, 2, \ldots\}$, the set of all natural numbers.
- A language is a set $L \subseteq \mathbb{N}$.
- A presentation for L is essentially an (infinite) listing T of all and only the elements of L. Such a T is called a text for L.
- We numerically name programs or grammars in some standard general hypothesis space, where each $e \in \mathbb{N}$ generates some language.

Success: TxtEx-Learning

- Let L be a language, h an algorithmic learner and T a text (a presentation) for L.
- For all k, we write $T[k]$ for the sequence $T(0), \ldots, T(k-1)$.
- The learning sequence p_T of h on T is given by

$$\forall k : p_T(k) = h(T[k]).$$

- Gold 1967: h TxtEx-learns L iff, for all texts T for L, there is i such that $p_T(i) = p_T(i+1) = p_T(i+2) = \ldots$ and $p_T(i)$ is a program for L.
- A class $\mathcal{L}$ of languages is TxtEx-learnable iff there exists an algorithmic learner h TxtEx-learning each language $L \in \mathcal{L}$.

Success: TxtEx-Learning

- Let L be a language, h an algorithmic learner and T a text (a presentation) for L.
- For all k, we write $T[k]$ for the sequence $T(0), \ldots, T(k-1)$.
- The learning sequence p_T of h on T is given by

$$\forall k : p_T(k) = h(T[k]).$$

- Gold 1967: h TxtEx-learns L iff, for all texts T for L, there is i such that $p_T(i) = p_T(i+1) = p_T(i+2) = \ldots$ and $p_T(i)$ is a program for L.
- A class $\mathcal{L}$ of languages is TxtEx-learnable iff there exists an algorithmic learner h TxtEx-learning each language $L \in \mathcal{L}$.

Success: TxtEx-Learning

- Let L be a language, h an algorithmic learner and T a text (a presentation) for L.
- For all k, we write $T[k]$ for the sequence $T(0), \ldots, T(k-1)$.
- The learning sequence p_T of h on T is given by

$$\forall k : p_T(k) = h(T[k]).$$

- Gold 1967: h TxtEx-learns L iff, for all texts T for L, there is i such that $p_T(i) = p_T(i+1) = p_T(i+2) = \ldots$ and $p_T(i)$ is a program for L.
- A class $\mathcal{L}$ of languages is TxtEx-learnable iff there exists an algorithmic learner h TxtEx-learning each language $L \in \mathcal{L}$.

Success: TxtEx-Learning

- Let L be a language, h an algorithmic learner and T a text (a presentation) for L.
- For all k, we write $T[k]$ for the sequence $T(0), \ldots, T(k-1)$.
- The learning sequence p_T of h on T is given by

$$\forall k : p_T(k) = h(T[k]).$$

- Gold 1967: h TxtEx-learns L iff, for all texts T for L, there is i such that $p_T(i) = p_T(i+1) = p_T(i+2) = \ldots$ and $p_T(i)$ is a program for L.
- A class $\mathcal{L}$ of languages is TxtEx-learnable iff there exists an algorithmic learner h TxtEx-learning each language $L \in \mathcal{L}$.

Success: TxtEx-Learning

- Let L be a language, h an algorithmic learner and T a text (a presentation) for L.
- For all k, we write $T[k]$ for the sequence $T(0), \ldots, T(k-1)$.
- The learning sequence p_T of h on T is given by

$$\forall k : p_T(k) = h(T[k]).$$

- Gold 1967: h TxtEx-learns L iff, for all texts T for L, there is i such that $p_T(i) = p_T(i+1) = p_T(i+2) = \ldots$ and $p_T(i)$ is a program for L.
- A class $\mathcal{L}$ of languages is TxtEx-learnable iff there exists an algorithmic learner h TxtEx-learning each language $L \in \mathcal{L}$.

Success: TxtEx-Learning

- Let L be a language, h an algorithmic learner and T a text (a presentation) for L.
- For all k, we write $T[k]$ for the sequence $T(0), \dots, T(k-1)$.
- The learning sequence p_T of h on T is given by

$$\forall k : p_T(k) = h(T[k]).$$

- Gold 1967: h TxtEx-learns L iff, for all texts T for L, there is i such that $p_T(i) = p_T(i+1) = p_T(i+2) = \dots$ and $p_T(i)$ is a program for L.
- A class $\mathcal{L}$ of languages is TxtEx-learnable iff there exists an algorithmic learner h TxtEx-learning each language $L \in \mathcal{L}$.

Success: TxtEx-Learning

- Let L be a language, h an algorithmic learner and T a text (a presentation) for L.
- For all k, we write $T[k]$ for the sequence $T(0), \ldots, T(k-1)$.
- The learning sequence p_T of h on T is given by

$$\forall k : p_T(k) = h(T[k]).$$

- Gold 1967: h TxtEx-learns L iff, for all texts T for L, there is i such that $p_T(i) = p_T(i+1) = p_T(i+2) = \ldots$ and $p_T(i)$ is a program for L.
- A class $\mathcal{L}$ of languages is TxtEx-learnable iff there exists an algorithmic learner h TxtEx-learning each language $L \in \mathcal{L}$.

Restrictions

- An (algorithmic) learner h is called set-driven iff, for all σ, τ listing the same (finite) set of elements, $h(\sigma) = h(\tau)$.
- A learner h is called partially set-driven iff, for all σ, τ of same length and listing the same set of elements, $h(\sigma) = h(\tau)$.

The above two restrictions model learner local-insensitivity to order of data presentation.

- A learner h is called iterative iff, for all σ, τ with $h(\sigma) = h(\tau)$, for all x, $h(\sigma \diamond x) = h(\tau \diamond x)$.[1]

[1] This is equivalent to a learner having access only to the current datum and the just prior hypothesis.

Restrictions

- An (algorithmic) learner h is called set-driven iff, for all σ, τ listing the same (finite) set of elements, $h(\sigma) = h(\tau)$.
- A learner h is called partially set-driven iff, for all σ, τ of same length and listing the same set of elements, $h(\sigma) = h(\tau)$.

The above two restrictions model learner local-insensitivity to order of data presentation.

- A learner h is called iterative iff, for all σ, τ with $h(\sigma) = h(\tau)$, for all x, $h(\sigma \diamond x) = h(\tau \diamond x)$.[1]

[1] This is equivalent to a learner having access only to the current datum and the just prior hypothesis.

Restrictions

- An (algorithmic) learner h is called set-driven iff, for all σ, τ listing the same (finite) set of elements, $h(\sigma) = h(\tau)$.
- A learner h is called partially set-driven iff, for all σ, τ of same length and listing the same set of elements, $h(\sigma) = h(\tau)$.

The above two restrictions model learner local-insensitivity to order of data presentation.

- A learner h is called iterative iff, for all σ, τ with $h(\sigma) = h(\tau)$, for all x, $h(\sigma \diamond x) = h(\tau \diamond x)$.[1]

[1] This is equivalent to a learner having access only to the current datum and the just prior hypothesis.

Restrictions

- An (algorithmic) learner *h* is called set-driven iff, for all σ, τ listing the same (finite) set of elements, $h(\sigma) = h(\tau)$.
- A learner *h* is called partially set-driven iff, for all σ, τ of same length and listing the same set of elements, $h(\sigma) = h(\tau)$.

The above two restrictions model learner local-insensitivity to order of data presentation.

- A learner *h* is called iterative iff, for all σ, τ with $h(\sigma) = h(\tau)$, for all x, $h(\sigma \diamond x) = h(\tau \diamond x)$.[1]

[1] This is equivalent to a learner having access only to the current datum and the just prior hypothesis.

Restrictions

- An (algorithmic) learner h is called set-driven iff, for all σ, τ listing the same (finite) set of elements, $h(\sigma) = h(\tau)$.
- A learner h is called partially set-driven iff, for all σ, τ of same length and listing the same set of elements, $h(\sigma) = h(\tau)$.

The above two restrictions model learner local-insensitivity to order of data presentation.

- A learner h is called iterative iff, for all σ, τ with $h(\sigma) = h(\tau)$, for all x, $h(\sigma \diamond x) = h(\tau \diamond x)$.[1]

[1] This is equivalent to a learner having access only to the current datum and the just prior hypothesis.

U-Shapes

For learning with any of the above restrictions we investigate the necessity of (two kinds of) U-shapes.
U-shaped learning occurs empirically in human child development: learn, unlearn, relearn.

- A learner h is said to be non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never semantically abandons a correct hypothesis.
- A learner h is said to be strongly non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never syntactically abandons a correct hypothesis.

U-Shapes

For learning with any of the above restrictions we investigate the necessity of (two kinds of) U-shapes.

U-shaped learning occurs empirically in human child development: learn, unlearn, relearn.

- A learner h is said to be non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never semantically abandons a correct hypothesis.
- A learner h is said to be strongly non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never syntactically abandons a correct hypothesis.

U-Shapes

For learning with any of the above restrictions we investigate the necessity of (two kinds of) U-shapes.
U-shaped learning occurs empirically in human child development: learn, unlearn, relearn.

- A learner h is said to be non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never semantically abandons a correct hypothesis.
- A learner h is said to be strongly non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never syntactically abandons a correct hypothesis.

U-Shapes

For learning with any of the above restrictions we investigate the necessity of (two kinds of) U-shapes.
U-shaped learning occurs empirically in human child development: learn, unlearn, relearn.

- A learner h is said to be non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never semantically abandons a correct hypothesis.
- A learner h is said to be strongly non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never syntactically abandons a correct hypothesis.

U-Shapes

For learning with any of the above restrictions we investigate the necessity of (two kinds of) U-shapes.
U-shaped learning occurs empirically in human child development: learn, unlearn, relearn.

- A learner h is said to be non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never semantically abandons a correct hypothesis.
- A learner h is said to be strongly non-U-shaped on a class of languages $\mathcal{L}$ iff, for each language $L \in \mathcal{L}$, h, when learning L, never syntactically abandons a correct hypothesis.

Results

- For set-driven learning, we can assume strongly non-U-shaped learners.
- For partially set-driven learning, we can assume strongly non-U-shaped learners.
- Surprisingly, for iterative learning, we cannot assume strongly non-U-shaped learners.

From Case and Moelius 2007, we know that, for iterative learning, we can assume (not necessarily strongly) non-U-shaped learners.

Results

- For set-driven learning, we can assume strongly non-U-shaped learners.
- For partially set-driven learning, we can assume strongly non-U-shaped learners.
- Surprisingly, for iterative learning, we cannot assume strongly non-U-shaped learners.

From Case and Moelius 2007, we know that, for iterative learning, we can assume (not necessarily strongly) non-U-shaped learners.

Results

- For set-driven learning, we can assume strongly non-U-shaped learners.
- For partially set-driven learning, we can assume strongly non-U-shaped learners.
- Surprisingly, for iterative learning, we cannot assume strongly non-U-shaped learners.

From Case and Moelius 2007, we know that, for iterative learning, we can assume (not necessarily strongly) non-U-shaped learners.

Results

- For set-driven learning, we can assume strongly non-U-shaped learners.
- For partially set-driven learning, we can assume strongly non-U-shaped learners.
- Surprisingly, for iterative learning, we cannot assume strongly non-U-shaped learners.

From Case and Moelius 2007, we know that, for iterative learning, we can assume (not necessarily strongly) non-U-shaped learners.

Results

- For set-driven learning, we can assume strongly non-U-shaped learners.
- For partially set-driven learning, we can assume strongly non-U-shaped learners.
- Surprisingly, for iterative learning, we cannot assume strongly non-U-shaped learners.

From Case and Moelius 2007, we know that, for iterative learning, we can assume (not necessarily strongly) non-U-shaped learners.

Techniques

How did we get those results?

- For unnecessary U-shapes, we give a general scheme for how to remove them.
- We apply this scheme for both set-driven and partially set-driven learning.
- We use an different (self-referential or self-learning) approach for showing the necessity of U-shapes.

Techniques

How did we get those results?

- For unnecessary U-shapes, we give a general scheme for how to remove them.
- We apply this scheme for both set-driven and partially set-driven learning.
- We use an different (self-referential or self-learning) approach for showing the necessity of U-shapes.

Techniques

How did we get those results?

- For unnecessary U-shapes, we give a general scheme for how to remove them.
- We apply this scheme for both set-driven and partially set-driven learning.
- We use an different (self-referential or self-learning) approach for showing the necessity of U-shapes.

Techniques

How did we get those results?

- For unnecessary U-shapes, we give a general scheme for how to remove them.
- We apply this scheme for both set-driven and partially set-driven learning.
- We use an different (self-referential or self-learning) approach for showing the necessity of U-shapes.

Surprise re Self-Learning Technique

- We have a very general result employing self-learning classes of languages to completely epitomize or characterize any strict learning power difference between two learning criteria.
- Suppose $\mathcal{L}$ is a self-learning class for this result. Each language of $\mathcal{L}$ contains only programs which completely specify how the corresponding learner of $\mathcal{L}$ is to transform its data into output programs.
- This technique applies well beyond criteria featuring presence or absence of U-shapes.

Surprise re Self-Learning Technique

- We have a very general result employing self-learning classes of languages to completely epitomize or characterize any strict learning power difference between two learning criteria.
- Suppose $\mathcal{L}$ is a self-learning class for this result. Each language of $\mathcal{L}$ contains only programs which completely specify how the corresponding learner of $\mathcal{L}$ is to transform its data into output programs.
- This technique applies well beyond criteria featuring presence or absence of U-shapes.

Surprise re Self-Learning Technique

- We have a very general result employing self-learning classes of languages to completely epitomize or characterize any strict learning power difference between two learning criteria.
- Suppose $\mathcal{L}$ is a self-learning class for this result. Each language of $\mathcal{L}$ contains only programs which completely specify how the corresponding learner of $\mathcal{L}$ is to transform its data into output programs.
- This technique applies well beyond criteria featuring presence or absence of U-shapes.

Surprise re Self-Learning Technique

- We have a very general result employing self-learning classes of languages to completely epitomize or characterize any strict learning power difference between two learning criteria.
- Suppose $\mathcal{L}$ is a self-learning class for this result. Each language of $\mathcal{L}$ contains only programs which completely specify how the corresponding learner of $\mathcal{L}$ is to transform its data into output programs.
- This technique applies well beyond criteria featuring presence or absence of U-shapes.

Conclusion and Future Work

- We added to the picture regarding the necessity of U-shapes.
- In the future, we will try to get an even better understanding wrt the necessity of U-shapes for other learning criteria.
- Regarding self-learning classes of languages, we currently work on a considerable expansion of the surprising result that self-learning classes characterize learning power differences.

Conclusion and Future Work

- We added to the picture regarding the necessity of U-shapes.
- In the future, we will try to get an even better understanding wrt the necessity of U-shapes for other learning criteria.
- Regarding self-learning classes of languages, we currently work on a considerable expansion of the surprising result that self-learning classes characterize learning power differences.

Conclusion and Future Work

- We added to the picture regarding the necessity of U-shapes.
- In the future, we will try to get an even better understanding wrt the necessity of U-shapes for other learning criteria.
- Regarding self-learning classes of languages, we currently work on a considerable expansion of the surprising result that self-learning classes characterize learning power differences.

Conclusion and Future Work

- We added to the picture regarding the necessity of U-shapes.
- In the future, we will try to get an even better understanding wrt the necessity of U-shapes for other learning criteria.
- Regarding self-learning classes of languages, we currently work on a considerable expansion of the surprising result that self-learning classes characterize learning power differences.

Thank You.