BILL
RECORD THIS
LECTURE

September 9, 2021

Gen Sub Cipher: How to Really Crack

September 9, 2021

General Substitution Cipher

Def Gen Sub Cipher with perm f on $\{0, \ldots, 25\}$.

1. Encrypt via $x \rightarrow f(x)$.
2. Decrypt via $x \rightarrow f^{-1}(x)$.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

1. The **1-grams** of T are just the letters in T, counting repeats.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

1. The **1-grams** of T are just the letters in T, counting repeats.
2. The **2-grams** of T are just the contiguous pairs of letters in T, counting repeats. Also called **bigrams**.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

1. The **1-grams** of T are just the letters in T, counting repeats.
2. The **2-grams** of T are just the contiguous pairs of letters in T, counting repeats. Also called **bigrams**.
3. The **3-grams** of T you can guess. Also called **trigrams**.

Terminology: 1-Gram, 2-Gram, 3-Gram

Notation Let T be a text.

1. The **1-grams** of T are just the letters in T, counting repeats.
2. The **2-grams** of T are just the contiguous pairs of letters in T, counting repeats. Also called **bigrams**.
3. The **3-grams** of T you can guess. Also called **trigrams**.
4. One usually talks about the freq of n-grams.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

The following 1-grams occur 4 times: r,s,t.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

The following 1-grams occur 4 times: r,s,t.

The following 1-gram occurs 6 times: o.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

The following 1-grams occur 4 times: r,s,t.

The following 1-gram occurs 6 times: o.

The following 1-gram occurs 9 times: e.

Example of 1-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 1-grams occur 1 time: a,d,u,v,y.

The following 1-grams occur 2 times: h,l,n,p,w.

The following 1-grams occur 3 times: c,i,m.

The following 1-grams occur 4 times: r,s,t.

The following 1-gram occurs 6 times: o.

The following 1-gram occurs 9 times: e.

No 1-gram occurs ≥ 10 times.

Example of 2-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

Example of 2-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 2-grams occur 2 times: me, or.

Example of 2-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 2-grams occur 2 times: me, or.

The following 2-grams occur 1 time: ev, ve, er, rn, no, ot, ti, ic, eh, ho, ow, ws, so, et, ti, im, es, sp, pe, eo, op, pl, le, eu, us, se, em, ma, at, th, hw, wo, ds, in, nc, co, rr, re, ec, ct, tl, ly.

Example of 2-Grams

Let the text be:

Ever notice how sometimes people use math words incorrectly?

The following 2-grams occur 2 times: me, or.

The following 2-grams occur 1 time: ev, ve, er, rn, no, ot, ti, ic, eh, ho, ow, ws, so, et, ti, im, es, sp, pe, eo, op, pl, le, eu, us, se, em, ma, at, th, hw, wo, ds, in, nc, co, rr, re, ec, ct, tl, ly.

No 2-gram occurs ≥ 3 times.

BILL: BEGIN HERE. RECORD THIS LECTURE

September 9, 2021

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. $f_{E,n}$ is freq of n-grams in English. It is a 26^n long vector.

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. $f_{E,n}$ is freq of n-grams in English. It is a 26^n long vector.
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. $f_{E,n}$ is freq of n-grams in English. It is a 26^n long vector.
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!
3. $f_{\sigma(T),n}$ is the 26^n-long vector of freq's of n-grams in $\sigma(T)$.

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. $f_{E,n}$ is freq of n-grams in English. It is a 26^n long vector.
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!
3. $f_{\sigma(T),n}$ is the 26^n-long vector of freq's of n-grams in $\sigma(T)$.
4. I and R will be parameters we discuss later.

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. $f_{E,n}$ is freq of n-grams in English. It is a 26^n long vector.
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!
3. $f_{\sigma(T),n}$ is the 26^n-long vector of freq's of n-grams in $\sigma(T)$.
4. I and R will be parameters we discuss later.
   I stands for $\mathrm{Iterations}$ and will be large (like 2000).

Notation and Parameter for a Family of Algorithms

Notation Let σ be a perm and T a text.

1. $f_{E,n}$ is freq of n-grams in English. It is a 26^n long vector.
2. $\sigma(T)$ is taking T and applying σ to it. If σ^{-1} was used to encrypt, then $\sigma(T)$ will be English!
3. $f_{\sigma(T),n}$ is the 26^n-long vector of freq's of n-grams in $\sigma(T)$.
4. I and R will be parameters we discuss later.
   I stands for $\mathrm{Iterations}$ and will be large (like 2000).
   R stands for Redos and will be small (like 5).

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

1. 1-grams: $f_{E,1} \cdot f_{E,1} \sim 0.065$.

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

1. 1-grams: $f_{E,1} \cdot f_{E,1} \sim 0.065$.
2. 2-grams: $f_{E,2} \cdot f_{E,2} \sim 0.0067$.

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

1. 1-grams: $f_{E,1} \cdot f_{E,1} \sim 0.065$.
2. 2-grams: $f_{E,2} \cdot f_{E,2} \sim 0.0067$.
3. 3-grams: $f_{E,3} \cdot f_{E,3} \sim 0.0011$.

Stats for 1-Gram, 2-Gram, 3-Gram, 4-Gram

1. 1-grams: $f_{E,1} \cdot f_{E,1} \sim 0.065$.
2. 2-grams: $f_{E,2} \cdot f_{E,2} \sim 0.0067$.
3. 3-grams: $f_{E,3} \cdot f_{E,3} \sim 0.0011$.
4. 4-grams: $f_{E,4} \cdot f_{E,4} \sim 0.00023$.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct shift. Large ~ 0.065.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect shift. Small ~ 0.035.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct perm. Large ~ 0.065.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct perm. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect perm. Small.

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct perm. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect perm. Small. Hmmm?

Contrast Shift to Gen Sub

To crack shift went through all 26 shifts σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct shift. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect shift. Small ~ 0.035.
3. **Important** Will always be large or small. So we have **a gap**.

Lets try this with gen sub, ignoring the issue of 26! perms.
To crack gen sub shift went through all 26! perm σ:

1. If $f_{\sigma(T),1} \cdot f_{E,1}$ is large then σ is correct perm. Large ~ 0.065.
2. If $f_{\sigma(T),1} \cdot f_{E,1}$ is small then σ is incorrect perm. Small. Hmmm?
3. We have a problem. If σ only changed a few letters around, then likely $f_{E,1} \cdot f_{\sigma(T),1}$ will be large. We **do not** have a gap!

What to do?

What to do if there is no Gap?

What to do if there is no Gap?

1. Use n-grams instead of 1-grams.

What to do if there is no Gap?

1. Use n-grams instead of 1-grams.
2. If σ is a perm and $n \in \mathbb{N}$ then

$$\mathrm{good}_{\sigma,n} = f_{E,n} \cdot f_{\sigma(T),n}.$$

What to do if there is no Gap?

1. Use n-grams instead of 1-grams.
2. If σ is a perm and $n \in \mathbb{N}$ then

$$\mathrm{good}_{\sigma,n} = f_{E,n} \cdot f_{\sigma(T),n}.$$

3. Rather than view the **Is-English** program as a YES-NO, view it as comparative:

   T_1 looks **more like English** than T_2.

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
σ_{init} is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\rm init}$

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to $\rm R$ ($\rm R$ is small, about 5)

> $\sigma_r \leftarrow \sigma_{\rm init}$
> For $i = 1$ to $\rm I$ ($\rm I$ is large, about 2000)

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to $\rm R$ ($\rm R$ is small, about 5)

  $\sigma_r \leftarrow \sigma_{\rm init}$
  For $i = 1$ to $\rm I$ ($\rm I$ is large, about 2000)
    Pick $j, k \in \{0, \ldots, 25\}$ at Random.

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
σ_{init} is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

  $\sigma_r \leftarrow \sigma_{\mathrm{init}}$
  For $i = 1$ to I (I is large, about 2000)
    Pick $j, k \in \{0, \ldots, 25\}$ at Random.
    Let σ' be σ_r with j, k swapped

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
σ_{init} is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

> $\sigma_r \leftarrow \sigma_{\text{init}}$
> For $i = 1$ to I (I is large, about 2000)
>
> > Pick $j, k \in \{0, \ldots, 25\}$ at Random.
> > Let σ' be σ_r with j, k swapped
> > If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
$\sigma_{\rm init}$ is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

- $\sigma_r \leftarrow \sigma_{\rm init}$
- For $i = 1$ to I (I is large, about 2000)
  - Pick $j, k \in \{0, \ldots, 25\}$ at Random.
  - Let σ' be σ_r with j, k swapped
  - If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$

n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.

σ_{init} is perm that maps most freq to e, etc. Uses 1-gram freq.

For $r = 1$ to R (R is small, about 5)

> $\sigma_r \leftarrow \sigma_{\mathrm{init}}$
>
> For $i = 1$ to I (I is large, about 2000)
>
> > Pick $j, k \in \{0, \ldots, 25\}$ at Random.
> > Let σ' be σ_r with j, k swapped
> > If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$

Pick the σ_r with max $\mathrm{good}_{\sigma_r,n}$ or have human look at all $\sigma_r(T)$

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

We Trebekked It!

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

We Trebekked It!
On the TV show JEOPARDY Alex Trebek (before he died) gives you the **answer** and you have to figure out the **question**.

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

We Trebekked It!
On the TV show JEOPARDY Alex Trebek (before he died) gives you the **answer** and you have to figure out the **question**.

Same here.

Finding Parameters: A Chicken-and-Egg Problem

An old question:
What came first, the chicken or the egg?

Our Problem We need parameters I and R so the answer looks like English. But we then need a notion of **Is English** that does not use a gap. Need a program to tell us that it looks like English.

We Trebekked It!
On the TV show JEOPARDY Alex Trebek (before he died) gives you the **answer** and you have to figure out the **question**.

Same here.

We find the parameters for texts where we know the answers.

Finding the Parameters

Do the following a large number of times:

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)
4. Run the n-gram algorithm but with no bound on the number of iterations. Stop when either

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)
4. Run the n-gram algorithm but with no bound on the number of iterations. Stop when either
   4.1 Get original text T, or

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)
4. Run the *n*-gram algorithm but with no bound on the number of iterations. Stop when either
    4.1 Get original text T, or
    4.2 Swaps do not improve how close to English (could be in local max). In this case try again.

Finding the Parameters

Do the following a large number of times:

1. Take a text T of $\sim 10,000$ characters.
2. Take a random perm σ.
3. Compute $\sigma(T)$. (Note- We know σ and T)
4. Run the n-gram algorithm but with no bound on the number of iterations. Stop when either
   4.1 Get original text T, or
   4.2 Swaps do not improve how close to English (could be in local max). In this case try again.
5. Keep track of how how many iterations suffice and how many redos suffice.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

He used a Mac-Book Pro with 2.2 Ghz 6-core Intel Core i7 processor and 16 GB of RAM.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

He used a Mac-Book Pro with 2.2 Ghz 6-core Intel Core i7 processor and 16 GB of RAM.

My Point He used a computer that an ugrad can buy and use.

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

He used a Mac-Book Pro with 2.2 Ghz 6-core Intel Core i7 processor and 16 GB of RAM.

My Point He used a computer that an ugrad can buy and use.

He ran the program to find parameters on 150 texts of size approx 10,000 characters:

David Zhen Found the Parameters

UMCP ugrad CS major David Zhen worked with me on this over the summer.

The next three slides show the parameters he found.

He used a Mac-Book Pro with 2.2 Ghz 6-core Intel Core i7 processor and 16 GB of RAM.

My Point He used a computer that an ugrad can buy and use.

He ran the program to find parameters on 150 texts of size approx 10,000 characters:

For each text he generated many random perm and ran the algorithm.

Parameters for n-Grams

Parameters for n-Grams

1-grams Nothing worked

Parameters for n-Grams

1-grams Nothing worked
2-grams Nothing worked

Parameters for n-Grams

1-grams Nothing worked
2-grams Nothing worked
3-grams $I = 2000$, $R = 4$ worked. Took ≤ 2 minutes to crack.

Parameters for n-Grams

1-grams Nothing worked
2-grams Nothing worked
3-grams $I = 2000$, $R = 4$ worked. Took ≤ 2 minutes to crack.
4-grams $I = 2000$, $R = 8$, Took around 6 minutes to crack.

Parameters for n-Grams

1-grams Nothing worked
2-grams Nothing worked
3-grams $I = 2000$, $R = 4$ worked. Took ≤ 2 minutes to crack.
4-grams $I = 2000$, $R = 8$, Took around 6 minutes to crack.

So the winner is 3-grams, with $I = 2000$ and $R = 4$.

Parameters for n-Grams

1-grams Nothing worked
2-grams Nothing worked
3-grams $I = 2000$, $R = 4$ worked. Took ≤ 2 minutes to crack.
4-grams $I = 2000$, $R = 8$, Took around 6 minutes to crack.

So the winner is 3-grams, with $I = 2000$ and $R = 4$.

Can we do better than 2 minutes? Can we do something clever?

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
For $r = 1$ to R (R is small, about 5)

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\mathrm{init}}$ (Could do this more cleverly)

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\mathrm{init}}$ (Could do this more cleverly)
For $i = 1$ to I (I is large, about 2000)

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.

For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\mathrm{init}}$ (Could do this more cleverly)

For $i = 1$ to I (I is large, about 2000)

Pick $j, k \in \{0, \ldots, 25\}$ at ~~Random~~ Cleverly!

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.

For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\mathrm{init}}$ (Could do this more cleverly)

For $i = 1$ to I (I is large, about 2000)

Pick $j, k \in \{0, \ldots, 25\}$ at ~~Random~~ Cleverly!

Let σ' be σ_r with j, k swapped

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
For $r = 1$ to R (R is small, about 5)

- $\sigma_r \leftarrow \sigma_{\mathrm{init}}$ (Could do this more cleverly)
- For $i = 1$ to I (I is large, about 2000)
  - Pick $j, k \in \{0, \ldots, 25\}$ at ~~Random~~ Cleverly!
  - Let σ' be σ_r with j, k swapped
  - If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.

For $r = 1$ to R (R is small, about 5)

> $\sigma_r \leftarrow \sigma_{\mathrm{init}}$ (Could do this more cleverly)
>
> For $i = 1$ to I (I is large, about 2000)
>
> > Pick $j, k \in \{0, \ldots, 25\}$ at ~~Random~~ Cleverly!
> >
> > Let σ' be σ_r with j, k swapped
> >
> > If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\text{init}}$ (Could do this more cleverly)
For $i = 1$ to I (I is large, about 2000)

Pick $j, k \in \{0, \ldots, 25\}$ at ~~Random~~ Cleverly!
Let σ' be σ_r with j, k swapped
If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$
Pick the σ_r with max $\mathrm{good}^n_{\sigma_r}$ or have human look at all $\sigma_r(T)$

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\rm init}$ (Could do this more cleverly)
For $i = 1$ to I (I is large, about 2000)

Pick $j, k \in \{0, \ldots, 25\}$ at ~~Random~~ Cleverly!
Let σ' be σ_r with j, k swapped
If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$
Pick the σ_r with max $\mathrm{good}^n_{\sigma_r}$ or have human look at all $\sigma_r(T)$

Tradeoff Lets say this takes less iterations. But we spend more time finding the clever swap. Is it worth it? Only way to find out is to DO IT.

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\mathrm{init}}$ (Could do this more cleverly)
For $i = 1$ to I (I is large, about 2000)

Pick $j, k \in \{0, \ldots, 25\}$ at ~~Random~~ Cleverly!
Let σ' be σ_r with j, k swapped
If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$
Pick the σ_r with max $\mathrm{good}^n_{\sigma_r}$ or have human look at all $\sigma_r(T)$

Tradeoff Lets say this takes less iterations. But we spend more time finding the clever swap. Is it worth it? Only way to find out is to DO IT.

A High School Student did this for me and claims it worked better- could use 400 instead of 2000 and it is faster.

A Possible Improvement to n-Gram Algorithm

Input T. Find Freq of 1-grams and n-grams.
For $r = 1$ to R (R is small, about 5)

$\sigma_r \leftarrow \sigma_{\mathrm{init}}$ (Could do this more cleverly)
For $i = 1$ to I (I is large, about 2000)

Pick $j, k \in \{0, \ldots, 25\}$ at ~~Random~~ Cleverly!
Let σ' be σ_r with j, k swapped
If $f_{\sigma'(T),n} \cdot f_{E,n} > f_{\sigma_r(T),n} \cdot f_{E,n}$ then $\sigma_r \leftarrow \sigma'$

Candidates for σ are $\sigma_1, \ldots, \sigma_{\mathrm{R}}$
Pick the σ_r with max $\mathrm{good}^n_{\sigma_r}$ or have human look at all $\sigma_r(T)$

Tradeoff Lets say this takes less iterations. But we spend more time finding the clever swap. Is it worth it? Only way to find out is to DO IT.

A High School Student did this for me and claims it worked better- could use 400 instead of 2000 and it is faster.
There were issues with his work so I would want to see this redone more carefully. However, I suspect

BILL
STOP RECORDING
THIS LECTURE

September 9, 2021