

# Programación en Python

---

ESTEBAN VÁZQUEZ

- 
1. Conceptos básicos de programación en Python
  2. Bucles y estructuras de decisión en Python
  3. Funciones y estructuras de datos en Python
  4. Uso de módulos y paquetes en Python
  5. Programación orientada a objetos en Python
  6. Manipulación de ficheros en Python.
  7. Estructura datos complejos.
  8. Data Wrangling y Cleansing con Python.
-

---

## Extras

1. Numpy.
  2. Pandas.
  3. Matplotlib.
  4. Dask.
  5. Scikit-Learn
-



---

# Conceptos básicos de programación en Python

## Tema 1



# ¿ Qué es un LP ?

---

Definición: Un lenguaje de programación es un sistema notacional para describir computaciones de una forma legible tanto para la máquina como para el ser humano.

Computación

Máquina Turing, tesis de Church

Legibilidad por parte de la máquina

Legibilidad por parte del ser humano

---



# Características

---

- Eficiencia
- Expresividad
- Capacidad de mantenimiento
- Legibilidad
- Confiabilidad
- Seguridad
- Simplicidad
- Productividad

---



# Abstracciones

---

Clases: Datos y control

Niveles: básicas, estructuradas y unitarias

## Abstracciones de datos:

- Básicas: tipos básicos (enteros, reales, ...)
- Estructuradas: tipos estructurados (arreglos, registros)
- Unitarias: Tipos abstractos de datos (TDAs), paquetes, módulos, clases, componentes



# Abstracciones

---

## Abstracciones de control

Básicas: asignación, goto

Estructuradas: condicionales e iteradores

Unitarias: paquetes, módulos, hilos y tareas.

Un lenguaje de programación es completo en Turing siempre que tenga variables enteras y aritméticas, y que ejecute enunciados en forma secuencial, incluyendo enunciados de asignación, selección e iteración.



# Paradigmas de programación

---

## Imperativo

modelo de Von Neuman, cuello de botella de Von Neuman

## Orientado a Objetos

TDAs, encapsulación, modularidad, reutilización

## Funcional

noción abstracta de función, cálculo lambda, recursividad, listas

## Lógico

Lógica simbólica, programación declarativa

---



# Definición del lenguaje

---

Sintaxis (estructura)

Gramáticas libres de contexto, estructura léxica, tokens

Semántica (significado)

Lenguaje natural

Semántica operacional

Semántica denotacional



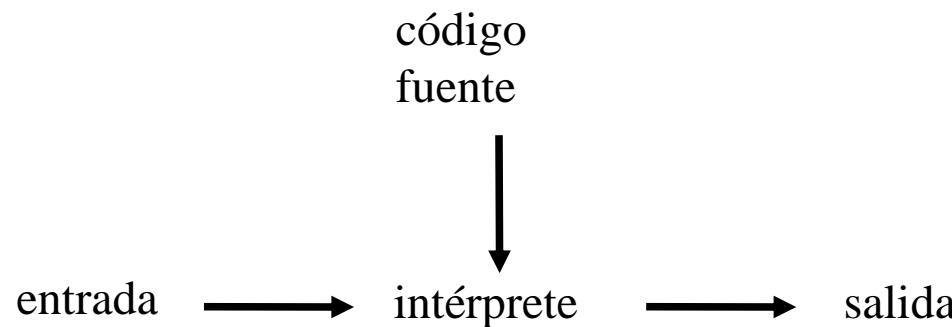
# Traducción del lenguaje

---

Traductor es un programa que acepta otros programas escritos en un lenguaje y:

los ejecuta directamente (interprete)

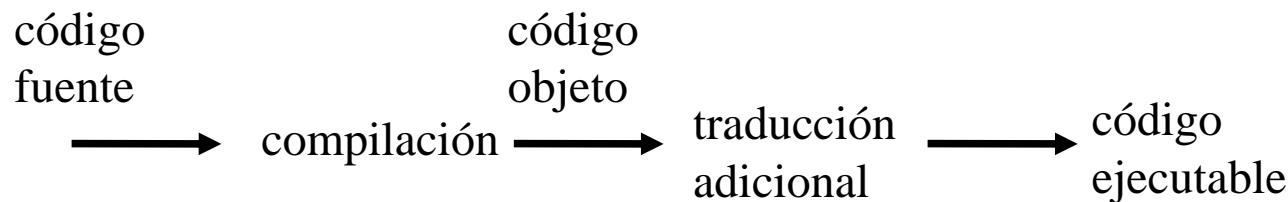
los transforma en una forma adecuada para su ejecución  
(compilador).





# Traducción

---



Pseudointérpretes: intermedio entre interprete y compilador:  
lenguajes intermedios

Operaciones de un traductor: analizador léxico (tokens),  
analizador sintáctico, analizador semántico, preprocesador



# Traducción

---

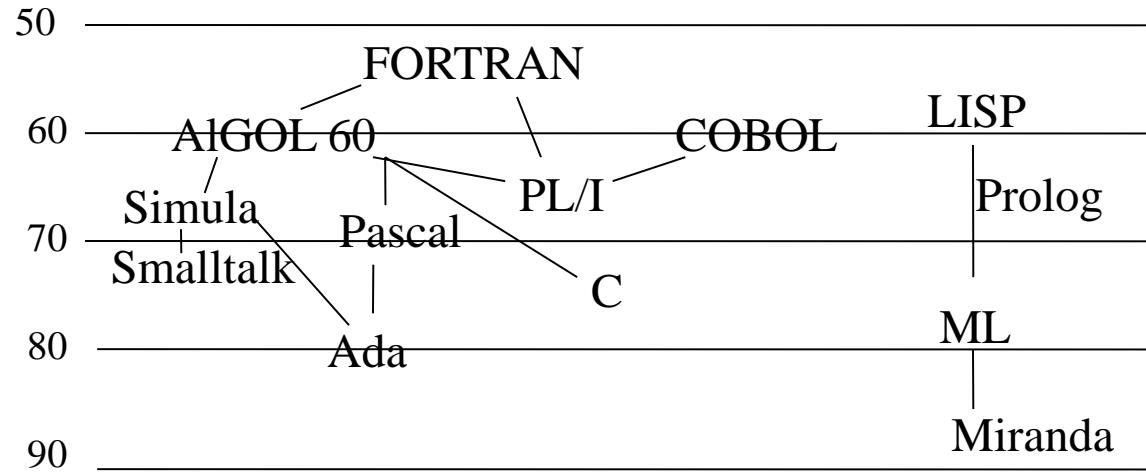
Tiempo de compilación y tiempo de ejecución

Propiedades estáticas: tiempo de compilación

Propiedades dinámicas: tiempo de ejecución

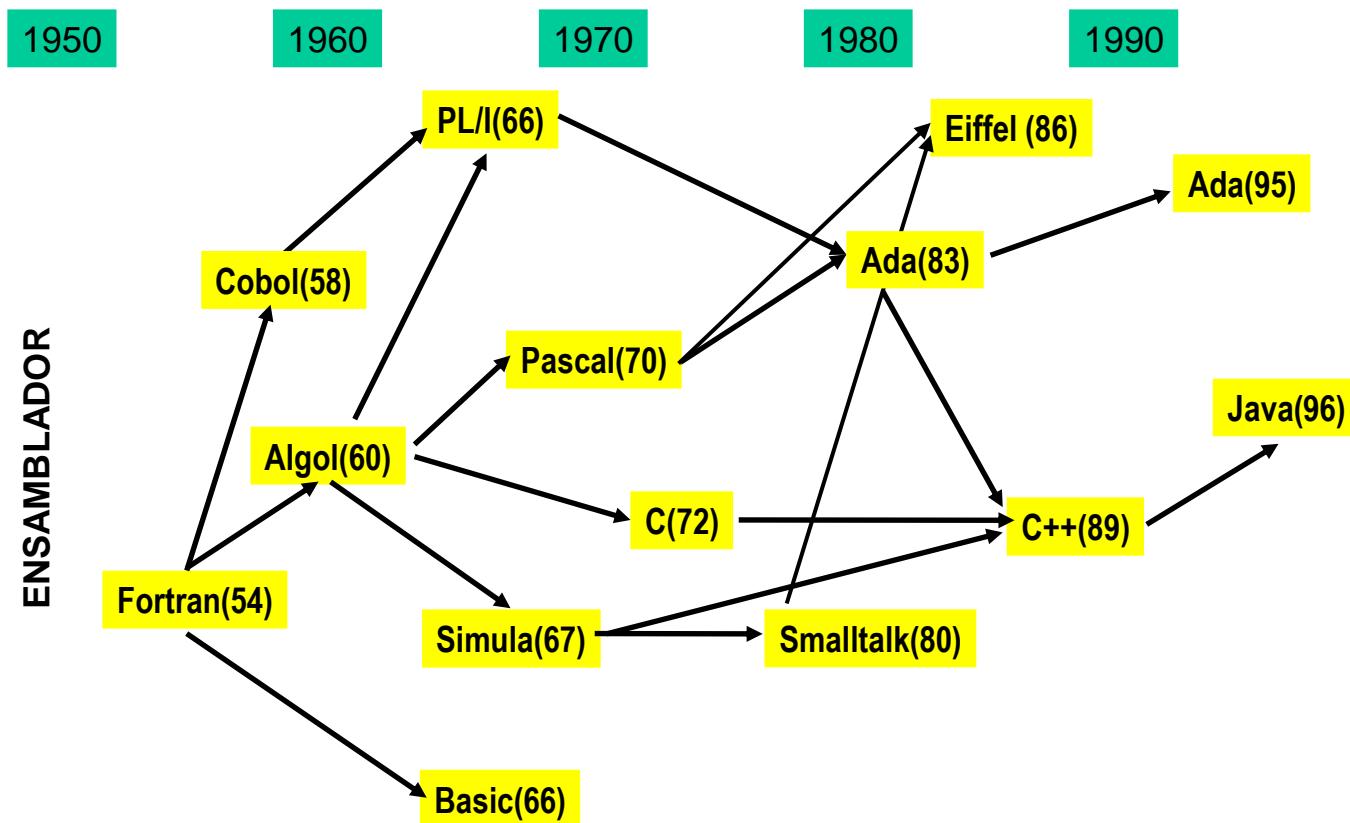
Recuperación de errores (compilación y ejecución)

Eficiencia y optimización (compilación o ejecución)





# Historia





# Preguntas

---

Clasifica los siguientes lenguajes (C, Ada, C++, Java, LISP, Prolog, Visual Basic, JavaScript, C#, PHP) en base a :

Paradigma

tipos de traductor (compilador, interprete, pseudointérprete)

Propiedades estáticas y dinámicas

Eficiencia

Extraed conclusiones de esta clasificación

---



# ¿Qué es Python?

---

- Python es un lenguaje de programación interpretado de alto nivel y multiplataforma (Windows, MacOS, Linux). Creado por [Guido van Rossum](#) (1991).
  - Es sencillo de aprender y de entender.
  - Los archivos de python tienen la extensión .py
    - Archivos de texto que son interpretados por el compilador. Para ejecutar programas en Python es necesario el *intérprete de python*, y el código a ejecutar.
  - Python dispone de un entorno interactivo y muchos módulos para todo tipo de aplicaciones.
-



# Instalación de Python

---

- La última versión de Python es la 3.
- Sitio oficial de descargas.
  - Con ello se instala el intérprete Python, IDLE (Integrated Development and Learning Environment), and Tkinter.
  - Se recomienda incluir python en la variable de entorno PATH
- Sitio oficial de documentación



# Instalación de librerías científicas en Python

- Los módulos se instalan con el comando pip

```
> python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose
```

*Table I-1. List of Fields of Study and Corresponding Python Modules*

Field of Study	Name of Python Module
Scientific Computation	scipy, numpy, sympy
Statistics	pandas
Networking	networkx
Cryptography	pyOpenSSL
Game Development	PyGame
Graphic User Interface	pyQT
Machine Learning	scikit-learn, tensorflow
Image Processing	scikit-image
Plotting	Matplotlib
Database	SQLAlchemy
HTML and XML parsing	BeautifulSoup
Natural Language Processing	nltk
Testing	nose



# Distribuciones alternativas de Python

---

- Existen distribuciones alternativas de Python:
  - [IronPython](#) (Python running on .NET)
  - [Jython](#) (Python running on the Java Virtual Machine)
  - [PyPy](#) (A fast python implementation with a JIT compiler)
  - [Stackless Python](#) (Branch of CPython with microthreads)
  - [MicroPython](#) (Python running on micro controllers)
  - [IPython](#) (provides a rich architecture for interactive computing)



# Implementaciones alternativas de Python

---

- Hay paquetes que incluyen librerías especializadas:
  - [ActiveState ActivePython](#) (scientific computing modules)
  - [pythonxy](#) (Scientific-oriented Python Distribution)
  - [winpython](#) (scientific Python distribution for Windows)
  - [Conceptive Python SDK](#) (business, desktop and database)
  - [Enthought Canopy](#) (for scientific computing)
  - [PyIMSL Studio](#) (for numerical analysis)
  - [Anaconda](#) (for data management, analysis and visualization of large data sets)
  - [eGenix PyRun](#) (portable Python runtime)
- Versión cloud:
  - [PythonAnywhere](#) (run Python in the browser)



# Tipos de datos

---

- Un tipo de dato es el conjunto de valores y el conjunto de operaciones definidas en esos valores.
- Python tiene un gran número de tipos de datos incorporados tales como Números (Integer, Float, Boolean, Complex Number), String, List, Tuple, Set, Dictionary and File.
- Otros tipos de datos de alto nivel, tales como Decimal y Fraction, están soportados por módulos externos.



# Definiciones

---

- Objetos. Todos los datos en un programa Python se representan por objetos. Cada objeto se caracteriza por su identidad, tipo y valor.
  - Referencias a objetos.
  - Literales.
  - Operadores.
  - Identificadores.
  - Variables.
  - Expresiones.
-



# Palabras reservadas

- Las palabras reservadas no se pueden usar como identificadores.

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	



# Integers - Enteros

- Tipo de dato (`int`) para representar enteros o números naturales.

<i>values</i>	<i>integers</i>						
<i>typical literals</i>	1234	99	0	1000000			
<i>operations</i>	<i>sign</i>	<i>add</i>	<i>subtract</i>	<i>multiply</i>	<i>floored divide</i>	<i>remainder</i>	<i>power</i>
<i>operators</i>	<code>+</code>	<code>-</code>	<code>+</code>	<code>-</code>	<code>*</code>	<code>//</code>	<code>%</code>
<i>Python's int data type</i>							

- Se puede expresar enteros en hexadecimal con el prefijo `0x` (o `0X`); en octal con el prefijo `0o` (o `0O`); y en binario con prefijo `0b` (o `0B`). Ejemplo: `0x1abc`, `0X1ABC`, `0o1776`, `0b11000011`.
- Adiferencia de otros lenguajes, los enteros en Python son de tamaño ilimitado.



# Integers - Enteros

---

- **Ejemplo**

```
>>> 123 + 456 - 789
-210
>>> 123456789012345678901234567890 + 1
123456789012345678901234567891
>>> 1234567890123456789012345678901234567890 + 1
1234567890123456789012345678901234567891
>>> 2 ** 888      # Raise 2 to the power of 888
.....
>>> len(str(2 ** 888))  # Convert integer to string and get its length
268                         # 2 to the power of 888 has 268 digits
>>> type(123)      # Get the type
<class 'int'>
>>> help(int)       # Show the help menu for type int
```



# Floating-Point Numbers - Reales

- Tipo de dato (`float`) para representar números en punto flotantes, para uso en aplicaciones científicas o comerciales

<i>values</i>	<i>real numbers</i>				
<i>typical literals</i>	3.14159	6.022e23	2.0	1.4142135623730951	
<i>operations</i>	<i>addition</i>	<i>subtraction</i>	<i>multiplication</i>	<i>division</i>	<i>exponentiation</i>
<i>operators</i>	+	-	*	/	**

*Python's float data type*

- Para obtener el máximo valor entero usar `sys.float_info.max`
- Tienen una representación IEEE de 64 bits. Típicamente tienen 15-17 dígitos decimales de precisión



# Floating-Point Numbers - Reales

---

- Ejemplo

```
>>> 1.23 * -4e5  
-492000.0  
>>> type(1.2)          # Get the type  
<class 'float'>  
>>> import math         # Using the math module  
>>> math.pi  
3.141592653589793  
>>> import random        # Using the random module  
>>> random.random()     # Generate a random number in [0, 1)  
0.890839384187198
```



# Números complejos

- Tipo de dato (`complex`) para representar números complejos de la forma  $a + bj$

```
>>> x = 1 + 2j # Assign var x to a complex number
>>> x           # Display x
(1+2j)
>>> x.real      # Get the real part
1.0
>>> x.imag      # Get the imaginary part
2.0
>>> type(x)     # Get type
<class 'complex'>
>>> x * (3 + 4j) # Multiply two complex numbers
(-5+10j)
>>> z = complex(2, -3) # Assign a complex number
```



# Booleans

---

- Tipo de dato (`bool`) que tiene dos valores: `True` y `False`
- El entero `0`, un valor vacío (como una cadena vacía `" "`, `""`, lista vacía `[]`, tuple vacía `()`, diccionario vacío `{}`), y `None` es tratado como `False`; todo lo demás es tratado como `True`.
- Los Booleans se comportan como enteros en operaciones aritméticas con `1` para `True` y `0` para `False`.



# Booleans

---

- **Ejemplo**

```
>>> 8 == 8      # Compare  
True  
>>> 8 == 9  
False  
>>> type(True)  # Get type  
<class 'bool'>  
>>> bool(0)  
False  
>>> bool(1)  
True  
>>> True + 3  
4  
>>> False + 1  
1
```

---



## Otros tipos

---

- Otros tipos de números son proporcionados por módulos externos, como decimal y fraction

```
# floats are imprecise
```

```
>>> 0.1 * 3
```

```
0.30000000000000004
```

```
# Decimal are precise
```

```
>>> import decimal # Using the decimal module
```

```
>>> x = decimal.Decimal('0.1') # Construct a Decimal object
```

```
>>> x * 3 # Multiply with overloaded * operator
```

```
Decimal('0.3')
```

```
>>> type(x) # Get type
```

```
<class 'decimal.Decimal'>
```



# El valor None

- Python proporciona un valor especial llamado `None` que puede ser usado para inicializar un objeto (en OOP)

```
>>> x = None
>>> type(x)    # Get type
<class 'NoneType'>
>>> print(x)
None

# Use 'is' and 'is not' to check for 'None' value.
>>> print(x is None)
True
>>> print(x is not None)
False
```



# Tipado dinámico y operador asignación

---

- Python es tipado dinámico, esto es, asocia tipos con objetos en lugar de variables. Así, una variable no tiene un tipo fijo y se le puede asignar un objeto de cualquier tipo. Una variable solo proporciona una referencia a un objeto.
  - No es necesario declarar una variable. Una variable se crea automáticamente cuando un valor es asignado la primera vez, que enlaza el objeto a la variable. Se puede usar la función implícita `type(nombre_var)` para obtener el tipo de objeto referenciado por una variable.
-



# Tipado dinámico y operador asignación

- Ejemplo:

```
>>> x = 1          # Assign an int value to create variable x
>>> x            # Display x
1
>>> type(x)      # Get the type of x
<class 'int'>
>>> x = 1.0       # Re-assign x to a float
>>> x
1.0
>>> type(x)      # Show the type
<class 'float'>
>>> x = 'hello'   # Re-assign x to a string
>>> x
'hello'
>>> type(x)      # Show the type
<class 'str'>
>>> x = '123'     # Re-assign x to a string (of digits)
>>> x
'123'
>>> type(x)      # Show the type
<class 'str'>
```



# Conversión de tipo

- Se puede convertir tipos mediante las funciones integradas `int()`, `float()`, `str()`, `bool()`, etc.

```
>>> x = '123'  
>>> type(x)  
<class 'str'>  
>>> x = int(x)      # Parse str to int, and assign back to x  
>>> x  
123  
>>> type(x)  
<class 'int'>  
>>> x = float(x)    # Convert x from int to float, and assign back to x  
>>> x  
123.0  
>>> type(x)  
<class 'float'>
```



# Conversión de tipo

```
>>> x = str(x)      # Convert x from float to str, and assign back to x
>>> x
'123.0'
>>> type(x)
<class 'str'>
>>> len(x)          # Get the length of the string
5
>>> x = bool(x)    # Convert x from str to boolean, and assign back to x
>>> x
# Non-empty string is converted to True
True
>>> type(x)
<class 'bool'>
>>> x = str(x)      # Convert x from bool to str
>>> x
'True'
```



# El operador asignación (=)

---

- En Python no es necesario *declarar* las variables antes de usarlas. La asignación inicial crea la variable y enlaza el valor a la variable

```
>>> x = 8          # Create a variable x by assigning a value  
>>> x = 'Hello'    # Re-assign a value (of a different type) to x
```

```
>>> y              # Cannot access undefined (unassigned) variable  
NameError: name 'y' is not defined
```



del

---

- Se puede usar la instrucción del para eliminar una variable

```
>>> x = 8      # Create variable x via assignment
>>> x
8
>>> del x     # Delete variable x
>>> x
NameError: name 'x' is not defined
```



# Asignación por pares y en cadena

- La asignación es asociativa por la derecha

```
>>> a = 1    # Ordinary assignment
>>> a
1
>>> b, c, d = 123, 4.5, 'Hello' # assignment of 3 variables pares
>>> b
123
>>> c
4.5
>>> d
'Hello'
>>> e = f = g = 123      # Chain assignment
>>> e
123
>>> f
123
>>> g
123
```



# Operadores aritméticos

Operador	Descripción	Ejemplos
+	<b>Addition</b>	
-	<b>Subtraction</b>	
*	<b>Multiplication</b>	
/	<b>Float Division (returns a float)</b>	$1/2 \Rightarrow 0.5$ $-1/2 \Rightarrow -0.5$
//	<b>Integer Division (returns the floor integer)</b>	$1//2 \Rightarrow 0$ $-1//2 \Rightarrow -1$ $8.9//2.5 \Rightarrow 3.0$ $-8.9//2.5 \Rightarrow -4.0$ $-8.9// -2.5 \Rightarrow 3.0$
**	<b>Exponentiation</b>	$2^{**}5 \Rightarrow 32$ $1.2^{**}3.4 \Rightarrow 1.858729691979481$
%	<b>Modulus (Remainder)</b>	$9\%2 \Rightarrow 1$ $-9\%2 \Rightarrow 1$ $9\%-2 \Rightarrow -1$ $-9\%-2 \Rightarrow -1$ $9.9\%2.1 \Rightarrow 1.5$ $-9.9\%2.1 \Rightarrow 0.6000000000000001$



# Operadores de comparación

- Los operadores de comparación se aplican a enteros y flotantes y producen un resultado booleano

Operador	Descripción	Ejemplo	
<, <=, >, >=, ==, !=	<b>Comparison</b>	$2 == 3$	$3 != 2$
		$2 < 13$	$2 <= 2$
		$13 > 2$	$3 >= 3$
in, not in	<b>x in y</b> comprueba si x está contenido en la secuencia y	<code>lis =[1, 4, 3, 2, 5]</code> <code>if 4 in lis: ....</code> <code>if 4 not in lis: ...</code>	
is, is not	<b>x is y</b> es True si x y y hacen referencia al mismo objeto	<code>x = 5</code> <code>if (type(x) is int): ...</code> <code>x = 5.2</code> <code>if (type(x) is not int): ...</code>	



# Operadores lógicos

- Se aplican a booleans. No hay exclusive-or (xor)

Operador	Descripción
and	<b>Logical AND</b>
or	<b>Logical OR</b>
not	<b>Logical NOT</b>

a	not a	a	b	a and b	a or b
False	True	False	False	False	False
True	False	False	True	False	True
		True	False	False	True
		True	True	True	True

*Truth-table definitions of bool operations*



# Operadores de bits

- Permiten operaciones a nivel de bits

Operador	Descripción	Ejemplo $x=0b10000$ 001 $y=0b10001111$
&	<b>bitwise AND</b>	$x \& y \Rightarrow 0b10000001$
	<b>bitwise OR</b>	$x   y \Rightarrow 0b10001111$
~	<b>bitwise NOT (or negate)</b>	$\sim x \Rightarrow -0b10000010$
^	<b>bitwise XOR</b>	$x ^ y \Rightarrow 0b00001110$
<<	<b>bitwise Left-Shift (padded with zeros)</b>	$x << 2 \Rightarrow 0b1000000100$
>>	<b>bitwise Right-Shift (padded with zeros)</b>	$x >> 2 \Rightarrow 0b100000$



# Operadores de asignación

Operador	Ejemplo	Equivalente a
=	x=5	x=5
+=	x+=5	x=x+5
-=	x-=5	x=x- 5
*=	x *=5	x=x * 5
/=	x/=5	x=x/ 5
%=	x% =5	x=x%5
//=	x// =5	x=x// 5
**=	x **=5	x=x ** 5
&=	x &=5	x=x&5
=	x  =5	x=x  5
^=	x ^=5	x=x^ 5
>>=	x >>=5	x=x>>5
<<=	x <<=5	x=x<<5

b, c, d = 123, 4.5, 'Hello' # asignación multiple



# Funciones integradas

- Python contiene funciones integradas para manipular números:
  - Matemáticas: `round()`, `pow()`, `abs()`
  - Conversión de tipos: `int()`, `float()`, `str()`, `bool()`, `type()`
  - Conversión de base: `hex()`, `bin()`, `oct()`

```
>>> x = 1.23456 # Test built-in function round()
>>> type(x)
<type 'float'>
>>> round(x)      # Round to the nearest integer
1
>>> type(round(x))
<class 'int'>
```



# Funciones integradas

```
>>> round(x, 1) # Round to 1 decimal place  
1.2  
>>> round(x, 2) # Round to 2 decimal places  
1.23  
>>> round(x, 8) # Nochange - not for formatting  
1.23456  
>>> pow(2, 5) # Test other built-in functions  
32  
>>> abs(-4.1)  
4.1  
# Base radix conversion  
>>> hex(1234)  
'0x4d2'  
>>> bin(254)  
'0b11111110'  
>>> oct(1234)  
'0o2322'  

```



# Funciones integradas

---

```
# List built-in functions
>>> dir(__builtins__)
['type', 'round', 'abs', 'int', 'float', 'str', 'bool', 'hex',
'bin', 'oct',.....]

# Show number of built-in functions
>>> len(dir(__builtins__)) # Python 3
151

# Show documentation of __builtins__ module
>>> help(__builtins__)
```



# Cadenas de caracteres - Strings

---

- Tipo de dato (`str`) para representar cadenas de caracteres, para uso en procesado de textos.
  - Se delimitan por (...), ("..."), ("..."..."), o ("""..."""")
  - Python 3 usa el conjunto de caracteres Unicode
  - Para especificar caracteres especiales se usan “secuencias de escape”. Ejemplo: \t, \n, \r
  - Los String son immutables, es decir, su contenido no se puede modificar
  - Para convertir números en strings se usa la función `str()`
  - Para convertir strings a números se usa `int()` o `float()`



# Ejemplo Strings

---

```
>>> s1 = 'apple'  
>>> s1  
'apple'  
>>> s2 = "orange"  
>>> s2  
'orange'  
>>> s3 = "'orange'"      # Escape sequence not required  
>>> s3  
"'orange'"  
>>> s3 = "\"orange\""\n      # Escape sequence needed  
>>> s3  
"\"orange\""  
  
# A triple-single/double-quoted string can span multiple lines  
>>> s4 = """testing  
testing"""  
>>> s4  
'testing\ntesting'
```

---



# Funciones y operadores para cadenas de caracteres

Función/Operador	Descripción	Ejemplos <code>s = 'Hello'</code>
<code>len()</code>	<b>Length</b>	<code>len(s) ⇒ 5</code>
<code>in</code>	<b>Contain?</b>	<code>'ell' in s ⇒ True</code> <code>'he' in s ⇒ False</code>
<code>+</code>	<b>Concatenation</b>	<code>s + '!' ⇒ 'Hello!'</code>
<code>*</code>	<b>Repetition</b>	<code>s * 2 ⇒ 'HelloHello'</code>
<code>[i], [-i]</code>	<b>Indexing to get a character.</b> <b>The front index begins at 0; back index begins at -1 (<code>=len()-1</code>).</b>	<code>s[1] ⇒ 'e'</code> <code>s[-4] ⇒ 'e'</code>
<code>[m:n], [m:], [:n], [m:n:step]</code>	<b>Slicing to get a substring.</b> <b>From index m (included) to n (excluded) with an optional step size.</b> <b>The default m=0, n=-1, step=1.</b>	<code>s[1:3] ⇒ 'el'</code> <code>s[1:-2] ⇒ 'el'</code> <code>s[3:] ⇒ 'lo'</code> <code>s[:-2] ⇒ 'He'</code> <code>s[:] ⇒ 'Hello'</code> <code>s[0:5:2] ⇒ 'Hlo'</code>



# Ejemplo de funciones/operadores Strings

```
>>>s = "Hello, world"      # Assign a string literal to the variable  
s  
>>>type(s)                  # Get data type of s  
<class 'str'>  
>>>len(s)                   # Length  
12  
>>>'ello' in s    # The in operator  
True  
# Indexing  
>>>s[0]                      # Get character at index 0; index begins at 0  
'H'  
>>>s[1]  
'e'  
>>>s[-1]                     # Get Last character, same as s[len(s) - 1]  
'd'  
>>>s[-2]                     # 2nd last character  
'l'
```



# Ejemplo de funciones/operadores Strings

```
# Slicing
>>> s[1:3]          # Substring from index 1 (included) to 3 (excluded)
'el'
>>> s[1:-1]
'ello, worl'
>>> s[:4]           # Same as s[0:4], from the beginning
'Hello'
>>> s[4:]           # Same as s[4:-1], till the end
'o, world'
>>> s[:]             # Entire string; same as s[0:len(s)]
'Hello, world'
# Concatenation (+) and Repetition (*)
>>> s = s + " again"    # Concatenate two strings
>>> s
'Hello, world again'
>>> s * 3            # Repeat 3 times
'Hello, world againHello, world againHello, world again'
>>> s[0] = 'a'# String is immutable
TypeError: 'str' object does not support item assignment
```



# Funciones específicas para cadenas de caracteres

---

- La clase str proporciona varias funciones miembro.  
Suponiendo que s es un objeto str:
  - s.strip(), s.rstrip(), s.lstrip(): the strip() strips the leading and trailing whitespaces. The rstrip() strips the right (trailing) whitespaces; while lstrip() strips the left (leading) whitespaces.
  - s.upper(), s.lower(), s.isupper(), s.islower()
  - s.find(s), s.index(s)
  - s.startswith(s)
  - s.endswith(s)
  - s.split(delimiter-str), delimiter-str.join(list-of-strings)



# Conversión de tipos

- Explícita: uso de funciones `int()`, `float()`, `str()`, y `round()`

<i>function call</i>	<i>description</i>
<code>str(x)</code>	<i>conversion of object x to a string</i>
<code>int(x)</code>	<i>conversion of string x to an integer or conversion of float x to an integer by truncation towards zero</i>
<code>float(x)</code>	<i>conversion of string or integer x to a float</i>
<code>round(x)</code>	<i>nearest integer to number x</i>

*APIs for some built-in type conversion functions*

- Implícita: Python convierte automáticamente enteros y flotantes convenientemente.



# ¿Tipo caracter?

---

Python no tiene un tipo de dato dedicado a caracteres. Un carácter es un string de longitud 1. Las funciones integradas `ord()` y `char()` operan sobre string 1

```
>>> ord('A') # ord(c) returns the integer ordinal (Unicode)
65
>>> ord('水')
27700
# chr(i) returns a one-character string with Unicode ordinal i
# 0 <= i <= 0x10ffff.
>>> chr(65)
'A'
>>> chr(27700)
'水'
```



# Formato de Strings

## Python 3 usa la función `format()` y `{ }`

```
# Replace format fields {} by arguments in format() in the same order
```

```
>>> '|{}|{}|more|'.format('Hello', 'world')
'|Hello|world|more|'
```

```
# You can use positional index in the form of {0}, {1}, ...
```

```
>>> '|{0}|{1}|more|'.format('Hello', 'world')
'|Hello|world|more|
>>> '|{1}|{0}|more|'.format('Hello', 'world')
'|world>Hello|more|'
```

```
# You can use keyword inside { }
```

```
>>> '|{greeting}|{name}|'.format(greeting='Hello', name='Peter')
'|Hello|Peter|'
```



# Formato de Strings

---

```
# specify field width and alignment in the form of i:n or key:n, #
where i positional index, key keyword, and n field width.
>>> '|{1:8}|{0:7}|'.format('Hello', 'Peter')
'|Peter    |Hello   |'      # Default left-aligned
#>(right align), <(left align), -<(fill char)
>>> '|{1:8}|{0:>7}|{2:-<10}|'.format('Hello', 'Peter', 'again')
'|Peter    |  Hello|again----|'
>>> '|{greeting:8}|{name:7}|'.format(name='Peter', greeting='Hi')
'|Hi        |Peter   |'
# Format int using 'd' or 'nd', Format float using 'f' or 'n.mf'
>>> '|{0:.3f}|{1:6.2f}|{2:4d}|'.format(1.2, 3.456, 78)
'|1.200|  3.46|  78|'
# With keywords
>>> '|{a:.3f}|{b:6.2f}|{c:4d}|'.format(a=1.2, b=3.456, c=78)
'|1.200|  3.46|  78|'
```

---



# Formato de Strings

---

Se pueden usar las funciones `str.rjust(n)`,  
`str.ljust(n)`, `str.center(n)`, `str.zfill(n)`  
donde n es el ancho de campo

```
>>> '123'.rjust(5) # Setting field width and alignment '
'123'
>>> '123'.ljust(5)
'123 '
>>> '123'.center(5)
' 123 '
>>> '123'.zfill(5)   # Pad with leading zeros
'00123'
>>> '1.2'.rjust(5) # Floats
' 1.2'
>>> '-1.2'.zfill(6)
'-001.2'
```

---



# Listas

---

- Python dispone de una estructura de datos potente integrada (lista - list) para arrays dinámicos.
- Una lista es encerrada entre corchetes [ ].
- Puede contener elementos de diferentes tipos.
- Puede crecer y encogerse dinámicamente.
- Los elementos se acceden mediante índice, **empezando por cero**.
- Hay funciones integradas (ej. `len()` , `max()` , `min()`, `sum()`), y operadores.



# Operadores para listas

Operador	Descripción	Ejemplos lst = [8, 9, 6, 2]
in	<b>Contain?</b>	9 in lst ⇒ True 5 in lst ⇒ False
+	<b>Concatenation</b>	lst + [5, 2] ⇒ [8, 9, 6, 2, 5, 2]
*	<b>Repetition</b>	lst * 2 ⇒ [8, 9, 6, 2, 8, 9, 6, 2]
[i], [-i]	<b>Indexing to get an item.</b> <b>Front index begins at 0; back index begins at -1 (or len-1).</b>	lst[1] ⇒ 9 lst[-2] ⇒ 6 lst[1] = 99 ⇒ modify an existing item
[m:n], [m:], [:n], [m:n:step]	<b>Slicing to get a sublist.</b> <b>From index m (included) to n (excluded) with an optional step size.</b> <b>The default m is 0, n is len-1.</b>	lst[1:3] ⇒ [9, 6] lst[1:-2] ⇒ [9] lst[3:] ⇒ [2] lst[:-2] ⇒ [8, 9] lst[:] ⇒ [8, 9, 6, 2] lst[0:4:2] ⇒ [8, 6] newlst = lst[:] ⇒ copy the list lst[4:] = [1, 2] ⇒ modify a sub-list
del	<b>Delete one or more items (for mutable sequences only)</b>	del lst[1] ⇒ lst is [8, 6, 2] del lst[1:] ⇒ lst is [8] del lst[:] ⇒ lst is [] (clear all items)



# Funciones para listas

Función	Descripción	Ejemplos lst = [8, 9, 6, 2]
len()	<b>Length</b>	len(lst) ⇒ 4
max(), min()	<b>Maximum and minimum value (for list of numbers only)</b>	max(lst) ⇒ 9 min(lst) ⇒ 2
sum()	<b>Sum (for list of numbers only)</b>	sum(lst) ⇒ 16

- Suponiendo que lst es un objeto list:
  - lst.append(item): append the given item behind the lst and return None; same as lst[len(lst):] = [item].
  - lst.extend(lst2): append the given list lst2 behind the lst and return None; same as lst[len(lst):] = lst2.
  - lst.insert(index, item): insert the given item before the index and return None. Hence, lst.insert(0, item) inserts before the first item of the lst; lst.insert(len(lst), item) inserts at the end of the lst which is the same as lst.append(item).
  - lst.index(item): return the index of the first occurrence of item; or error.
  - lst.remove(item): remove the first occurrence of item from the lst and return None; or error.
  - lst.pop(): remove and return the last item of the lst.
  - lst.pop(index): remove and return the indexed item of the lst.
  - lst.clear(): remove all the items from the lst and return None; same as del lst[:].
  - lst.count(item): return the occurrences of item.
  - lst.reverse(): reverse the lst in place and return None.
  - lst.sort(): sort the lst in place and return None.
  - lst.copy(): return a copy of lst; same as lst[:].



# Tuplas

- Es similar a las listas excepto que es inmutable (como los string).
- Consiste en una serie de elementos separados por comas, encerrados entre paréntesis.
- Se puede convertir a listas mediante `list(tupla)`.
- Se opera sobre tuplas (`tup`) con:
  - funciones integradas `len(tup)`, para tuplas de números `max(tup)`, `min(tup)`, `sum(tup)`
  - operadores como `in`, `+` y `*`
  - funciones de tupla `tup.count(item)`, `tup.index(item)`, etc



# Diccionarios

---

- Soportan pares llave–valor (mappings). Es mutable.
- Un diccionario se encierra entre llaves { }. La llave y el valor se separa por : con el formato  
`{k1:v1, k2:v2, ...}`
- Adiferencia de las listas y tuplas que usan un índice entero para acceder a los elementos, los diccionarios se pueden indexar usando cualquier tipo llave (número, cadena, otros tipos).



# Ejemplo - Diccionarios

---

```
>>>dct ={'name':'Peter', 'gender':'male', 'age':21}
>>>dct
{'age': 21, 'name': 'Peter', 'gender': 'male'}
>>>dct['name']          # Get value via key
'Peter'
>>>dct['age'] = 22      # Re-assign a value
>>>dct
{'age': 22, 'name': 'Peter', 'gender': 'male'}
>>>len(dct)
3
>>>dct['email'] = 'pcmq@sant.com'    # Add new item
>>>dct
{'name': 'Peter', 'age': 22, 'email': 'pcmq@sant.com', 'gender':
'male'}
>>>type(dct)
<class 'dict'>
```

---



# Funciones para diccionarios

---

- Las más comunes son: (dct es un objeto dict)
  - dct.has\_key()
  - dct.items(), dct.keys(), dct.values()
  - dct.clear()
  - dct.copy()
  - dct.get()
  - dct.update(dct2): merge the given dictionary dct2 into dct.  
Override the value if key exists, else, add new key-value.
  - dct.pop()



# Operaciones comunes con diccionarios

## Common Dictionary Operations

Operation	Returns
<code>d = dict()</code> <code>d = dict(c)</code>	Creates a new empty dictionary or a duplicate copy of dictionary <i>c</i> .
<code>d = {}</code> <code>d = {k<sub>1</sub>: v<sub>1</sub>, k<sub>2</sub>: v<sub>2</sub>, ..., k<sub>n</sub>: v<sub>n</sub>}</code>	Creates a new empty dictionary or a dictionary that contains the initial items provided. Each item consists of a key ( <i>k</i> ) and a value ( <i>v</i> ) separated by a colon.
<code>len(d)</code>	Returns the number of items in dictionary <i>d</i> .
<code>key in d</code> <code>key not in d</code>	Determines if the key is in the dictionary.
<code>d[key] = value</code>	Adds a new <i>key/value</i> item to the dictionary if the <i>key</i> does not exist. If the key does exist, it modifies the value associated with the key.
<code>x = d[key]</code>	Returns the value associated with the given key. The key must exist or an exception is raised.



# Operaciones comunes con diccionarios

## Common Dictionary Operations

<code>d.get(key, default)</code>	Returns the value associated with the given key, or the default value if the key is not present.
<code>d.pop(key)</code>	Removes the key and its associated value from the dictionary that contains the given key or raises an exception if the key is not present.
<code>d.values()</code>	Returns a sequence containing all values of the dictionary.



# Conjuntos - set

---

- Es una colección de objetos sin ordenar no duplicados. Es una colección mutable, se puede usar add() para añadir elementos.
  - Un set se especifica encerrando los elementos entre entre llaves.
  - Se puede pensar que un set es un dict de llaves sin valor asociado.
  - Python tiene operadores set: & (intersection), | (union), - (difference), ^ (exclusive-or) y in (pertenencia).
-



# Operaciones comunes con conjuntos

## Common Set Operations

Operation	Description
<code>s = set()</code> <code>s = set(<i>seq</i>)</code> <code>s = {<i>e</i><sub>1</sub>, <i>e</i><sub>2</sub>, ..., <i>e</i><sub><i>n</i></sub>}</code>	Creates a new set that is either empty, a duplicate copy of sequence <i>seq</i> , or that contains the initial elements provided.
<code>len(s)</code>	Returns the number of elements in set <i>s</i> .
<code><i>element</i> in <i>s</i></code> <code><i>element</i> not in <i>s</i></code>	Determines if <i>element</i> is in the set.
<code>s.add(<i>element</i>)</code>	Adds a new element to the set. If the element is already in the set, no action is taken.
<code>s.discard(<i>element</i>)</code> <code>s.remove(<i>element</i>)</code>	Removes an element from the set. If the element is not a member of the set, <code>discard</code> has no effect, but <code>remove</code> will raise an exception.
<code>s.clear()</code>	Removes all elements from a set.
<code>s.issubset(<i>t</i>)</code>	Returns a Boolean indicating whether set <i>s</i> is a subset of set <i>t</i> .



# Operaciones comunes con conjuntos

## Common Set Operations

<code>s == t</code>	Returns a Boolean indicating whether set <i>s</i> is equal to set <i>t</i> .
<code>s != t</code>	
<code>s.union(<i>t</i>)</code>	Returns a new set that contains all elements in set <i>s</i> and set <i>t</i> .
<code>s.intersection(<i>t</i>)</code>	Returns a new set that contains elements that are in <i>both</i> set <i>s</i> and set <i>t</i> .
<code>s.difference(<i>t</i>)</code>	Returns a new set that contains elements in <i>s</i> that are not in set <i>t</i> .

Nota: `union`, `intersection` y `difference` devuelven nuevos conjuntos, no modifican el conjunto al que se aplica



# Estructuras complejas

---

- Los contenedores son muy útiles para almacenar colecciones de valores. Las listas y diccionarios pueden contener cualquier dato incluyendo otros contenedores.
- Así se puede crear un diccionario de conjuntos o diccionario de listas



# Funciones y APIs

- Tipos de funciones:  
integrad(a ( i n t () ,  
f l o a t () , s t r () ) ,  
standard o librería  
(math.sqrt()) requiere  
importar el módulo donde  
se encuentra.
- API: application  
programming interface

function call	description
<i>built-in functions</i>	
abs(x)	absolute value of x
max(a, b)	maximum value of a and b
min(a, b)	minimum value of a and b
<i>booksite functions for standard output from our stdio module</i>	
stdio.write(x)	write x to standard output
stdio.writeln(x)	write x to standard output, followed by a newline
<i>Note 1: Any type of data can be used (and will be automatically converted to str).</i>	
<i>Note 2: If no argument is specified, x defaults to the empty string.</i>	
<i>standard functions from Python's math module</i>	
math.sin(x)	sine of x (expressed in radians)
math.cos(x)	cosine of x (expressed in radians)
math.tan(x)	tangent of x (expressed in radians)
math.atan2(y, x)	polar angle of the point (x, y)
math.hypot(x, y)	Euclidean distance between the origin and (x, y)
math.radians(x)	conversion of x (expressed in degrees) to radians
math.degrees(x)	conversion of x (expressed in radians) to degrees
math.exp(x)	exponential function of x ( $e^x$ )
math.log(x, b)	base-b logarithm of x ( $\log_b x$ ) (the base b defaults to e—the natural logarithm)
math.sqrt(x)	square root of x
math.erf(x)	error function of x
math.gamma(x)	gamma function of x
<i>Note: The math module also includes the inverse functions asin(), acos(), and atan() and the constant variables e (2.718281828459045) and pi (3.141592653589793).</i>	
<i>standard functions from Python's random module</i>	
random.random()	a random float in the interval [0, 1)
random.randrange(x, y)	a random int in [x, y] where x and y are ints

APIs for some commonly used Python functions



---

# Bucles y estructuras de decisión en Python

## Tema 2



# Condicionales – if - else

- Se usa cuando se requiere realizar diferentes acciones para diferentes condiciones.
- Sintaxis general:

```
if test-1:  
    block-1  
elif test-2:  
    block-2  
....  
elif test-n:  
    block-n  
else:  
    else-block
```

Ejemplo:

```
if score >= 90:  
    letter = 'A'  
elif score >= 80:  
    letter = 'B'  
elif score >= 70:  
    letter = 'C'  
elif score >= 60:  
    letter = 'D'  
else:  
    letter = 'F'
```



# Operadores de comparación y lógicos

---

- Python dispone de operadores de comparación que devuelven un valor booleano True o False:
    - <, <=, ==, !=, >, >=
    - in, not in: Comprueba si un elemento está/no está en una secuencia (lista, tupla, etc).
    - is, is not: Comprueba si dos variables tienen la misma referencia
  - Python dispone de tres operadores lógicos (Boolean):
    - and
    - or
    - not
-



# Comparación encadenada

- Python permite comparación encadenada de la forma  
 $n_1 < x < n_2$

```
>>> x = 8
>>> 1 < x < 10
True
>>> 1 < x and x < 10 # Same as above
True
>>> 10 < x < 20
False
>>> 10 > x > 1
True
>>> not (10 < x < 20)
True
```



# Comparación de secuencias

- Los operadores de comparación están sobrecargados para aceptar secuencias (string, listas, tuplas)

```
>>> 'a' < 'b'
```

```
True
```

```
>>> 'ab' < 'aa'
```

```
False
```

```
>>> 'a' < 'b' < 'c'
```

```
True
```

```
>>> (1, 2, 3) < (1, 2, 4)
```

```
True
```

```
>>> [1, 2, 3] <= [1, 2, 3]
```

```
True
```



# Forma corta de if - else

---

- Sintaxis:

```
expr-1 if test else expr-2
```

```
# Evalua expr-1 si test es True; sino, evalua expr-2
```

```
>>> x = 0
```

```
>>> print('zero' if x == 0 else 'not zero')
```

```
zero
```

```
>>> x = -8
```

```
>>> abs_x = x if x > 0 else -x
```

```
>>> abs_x
```

```
8
```



## Ciclo while

- Instrucción que permite cálculos repetitivos sujetos a una condición. Sintaxis general:

```
while test:  
    true-block  
  
# while loop has an optional else block  
while test:  
    true-block  
else: # Run only if no break encounter ed  
    else-block
```

- El bloque `else` es opcional. Se ejecuta si se sale del ciclo sin encontrar una instrucción `break`.



## Ciclo while - Ejemplo

---

```
# Sum from 1 to the given upperbound
n = int(input('Enter      the upperbound: ' ))
i = 1
sum = 0
while ( i < n):
    sum += i
    i += 1
print(sum)
```



# Ciclo while - Ejemplo

```
import stdio
import sys
# Filename: powersoftwo.py. Accept positive integer n as a
# command-line argument. Write to standard output a table
# showing the first n powers of two.
n = int(sys.argv[1])
power = 1
i = 0
while i <= n:
    # Write the ith power of 2.
    print(str(i) + ' ' + str(power))
    power = 2 * power
    i = i + 1
# python powersoftwo.py 1
# 0 1
# 1 2
```



## Ciclos - for

- Sintaxis general del ciclo for - in:

```
# sequence:string,list,tuple,dictionary,set
for item in sequence:
    true-block
# for-in loop with a else block
for item in sequence:
    true-block
else:      # Run only if no break encounter ed
    else-block
```

- Se interpreta como “para cada ítem en la secuencia...”. El bloque else se ejecuta si el ciclo termina normalmente sin encontrar la instrucción break.



# Ciclos - for

---

- Ejemplos de iteraciones sobre una secuencia.

```
# String: iterating through each character
```

```
>>>for char in 'hello': print(char)
```

```
h  
e  
l  
l  
o
```

```
# List: iterating through each item
```

```
>>>for item in [123, 4.5, 'hello']: print(item)
```

```
123
```

```
4.5
```

```
Hello
```

```
# Tuple: iterating through each item
```

```
>>>for item in (123, 4.5, 'hello'): print(item)
```

```
123
```

```
4.5
```

```
hello
```



# Ciclos - for

---

```
# Dictionary: iterating through each key
```

```
>>>dct ={'a': 1, 2: 'b', 'c': 'cc'}  
>>>for key in dct: print(key, ':', dct[key])  
a: 1  
c: cc  
2: b
```

```
# Set: iterating through each item
```

```
>>>for item in {'apple', 1, 2, 'apple'}: print(item)  
1  
2  
apple
```

```
# File: iterating through each line
```

```
>>>f =open('test.txt', 'r')  
>>>for line in f: print(line)  
...Each line of the file...  
>>>f.close()
```

---



# Ciclos - for

- 
- Iteraciones sobre una secuencia de secuencias.

```
#A list of 2-item tuples
```

```
>>>lst =[(1,'a'), (2,'b'), (3,'c')]
```

```
#Iterating thru the each of the 2-item tuples
```

```
>>>for i1, i2 in lst: print(i1, i2)
```

```
...
```

```
1 a
```

```
2 b
```

```
3 c
```

```
#A list of 3-item lists
```

```
>>>lst =[[1, 2, 3], ['a', 'b', 'c']]
```

```
>>>for i1, i2, i3 in lst: print(i1, i2, i3)
```

```
...
```

```
1 2 3
```

```
a b c
```



# Ciclos - for

- 
- Iteraciones sobre un diccionario.

```
>>> dct = {'name':'Peter', 'gender':'male', 'age':21}
```

```
# Iterate through the keys (as in the above example)
```

```
>>> for key in dct: print(key, ':', dct[key])
```

```
age : 21
```

```
name: Peter
```

```
gender : male
```

```
# Iterate through the key-value pairs
```

```
>>> for key, value in dct.items(): print(key, ':', value)
```

```
age : 21
```

```
name: Peter
```

```
gender : male
```

```
>>> dct.items() # Return a list of key-value (2-item) tuples
```

```
[('gender', 'male'), ('age', 21), ('name', 'Peter')]
```



# Instrucción break

- `break` termina el ciclo y sigue en la instrucción que sigue al ciclo.

```
for var in sequence:  
    # codes inside for loop  
    if condition:  
        break  
    # codes inside for loop  
  
    ➔ # codes outside for loop
```

---

```
while test expression:  
    # codes inside while loop  
    if condition:  
        break  
    # codes inside while loop  
  
    ➔ # codes outside while loop
```



# Instrucción continue

- **continue** se usa para saltar el resto del código del ciclo y continuar con la siguiente iteración.

```
for var in sequence:  
    # codes inside for loop  
    if condition:  
        continue  
    # codes inside for loop  
  
    # codes outside for loop
```

---

```
while test expression:  
    # codes inside while loop  
    if condition:  
        continue  
    # codes inside while loop  
  
    # codes outside while loop
```



# Instrucciones pass, loop - else

---

- `pass` no hace nada. Sirve como marcador de una instrucción vacía o bloque vacío.
- `loop – else` se ejecuta si del ciclo se sale normalmente sin encontrar la instrucción `break`.



# Ciclos – for else

- Ejemplo de cláusula else en for

```
# List all primes between 2 and 100
for number in range(2, 101):
    for factor in range(2, number//2+1): # Look for factor
        if number % factor == 0: # break if a factor found
            print('%d is NOT a prime' % number)
            break
    else: # Only if no break encountered
        print('%d is a prime' % number)
```



---

# Funciones y estructuras de datos en Python

## Tema 3



# Funciones

- Se definen con la palabra clave `def` seguida por el nombre de la función, la lista de parámetros, las cadenas de documentación y el cuerpo de la función.
- Dentro del cuerpo de la función se puede usar la instrucción `return` para devolver un valor.
- Sintaxis:

```
def function_name(arg1, arg2, ...):
    """Function doc-string"""
    # Can be retrieved via function_name.__doc__
    # statements
    return return-value
```



# Funciones - Ejemplos

---

```
>>> def my_square(x):
    """Return the square of the given number"""
    return x * x

# Invoke the function defined earlier
>>> my_square(8)
64
>>> my_square(1.8)
3.24
>>> my_square('hello')
TypeError: can't multiply sequence by non-int of type
'str'
>>> my_square
<function my_square at 0x7fa57ec54bf8>
>>> type(my_square)
<class 'function'>
```



# Funciones - Ejemplos

---

```
>>> my_square.__doc__ # Show function doc-string  
'Return the square of the given number'  
>>> help(my_square) # Show documentation  
my_square(x)  
    Return the square of the given number  
>>> dir(my_square) # Show attributes  
[...]
```



# Funciones - Ejemplos

---

```
def fibon(n):
    """Print the first n Fibonacci numbers, where
       f(n)=f(n-1)+f(n-2) and f(1)=f(2)=1"""
    a, b = 1, 1
    for count in range(n):
        print(a, end=' ')  # print a space
        a, b = b, a+b
    print()  # print a newline

fibon(20)
```



# Funciones - Ejemplos

---

```
def my_cube(x):
    """( number) -> ( number)
    Return the cube of the given number.
    Examples (can be used by doctest):
    >>> my_cube(5)
    125
    >>> my_cube(-5)
    -125
    >>> my_cube(0)
    0
    """
    return x*x*x

# Test the function
print(my_cube(8))      # 512
print(my_cube(-8))     # -512
print(my_cube(0))       # 0
```

---



# Parámetros de funciones

---

- Los argumentos **inmutables** (enteros, floats, strings, tuplas) se pasan por *valor*. Es decir, se clona una copia y se pasa a la función, y el original no se puede modificar dentro de la función.
- Los argumentos **mutables** (listas, diccionarios, sets e instancias de clases) se pasan por *referencia*. Es decir, se pueden modificar dentro de la función.



# Parámetros de funciones con valores por defecto

---

- Se puede asignar un valor por defecto a los parámetros de funciones.

```
>>> def my_sum(n1, n2 = 4, n3 = 5): # n1 required, n2, n3 optional  
    """Return the sum of all the arguments"""  
    return n1 + n2 + n3
```

```
>>> print(my_sum(1, 2, 3))  
6  
>>> print(my_sum(1, 2))      # n3 defaults  
8  
>>> print(my_sum(1))        # n2 and n3 default  
10  
>>> print(my_sum())  
TypeError: my_sum() takes at least 1 argument (0 given)  
>>> print(my_sum(1, 2, 3, 4))  
TypeError: my_sum() takes at most 3 arguments (4 given)
```



# Argumentos posicionales y nominales

- Las funciones en Python permiten argumentos posicionales y nombrados.
- Normalmente se pasan los argumentos por posición de izquierda a derecha (posicional).

```
def my_sum(n1, n2 = 4, n3 = 5):
    """Return the sum of all the arguments"""
    return n1 + n2 + n3

print(my_sum(n2 = 2, n1 = 1, n3 = 3))
# Keyword arguments need not follow their positional order
print(my_sum(n2 = 2, n1 = 1))          # n3 defaults
print(my_sum(n1 = 1))                  # n2 and n3 default
print(my_sum(1, n3 = 3))              # n2 default
#print(my_sum(n2 = 2))                # TypeError, n1 missing
```



# Número de argumentos posicionales variables

---

- Python ofrece un número variable (arbitrario) de argumentos. En la definición de función se puede usar \* para indicar los restantes argumentos.

```
def my_sum(a, *args): # one posit.arg. & arbit.num.of args
    """Return the sum of all the arguments (one or more)"""
    sum=a
    for item in args: # args is a tuple
        sum += item
    return sum

print(my_sum(1))                  # args is ()
print(my_sum(1, 2))                # args is (2,)
print(my_sum(1, 2, 3))              # args is (2, 3)
print(my_sum(1, 2, 3, 4))            # args is (2, 3, 4)
```

---



# Número de argumentos posicionales variables

---

- Python permite poner `*args` en medio de la lista de parámetros. En ese caso todos los argumentos después de `*args` deben pasarse por nombre clave.

```
def my_sum( a, *args, b):  
    sum = a  
    for item in args:  
        sum += item  
    sum += b  
    return sum
```

```
print(my_sum(1, 2, 3, 4))  
#TypeError: my_sum() missing 1 required keyword-only argument: 'b'  
print(my_sum(1, 2, 3, 4, b=5))
```

---



# Número de argumentos posicionales variables

---

- De forma inversa cuando los argumentos ya están en una lista/tupla, se puede usar \* para desempacar la lista/tupla como argumentos posicionales separados.

```
>>> def my_sum(a, b, c): return a+b+c

>>> lst1 = [11, 22, 33]
# my_sum() expects 3 arguments, NOT a 3-item list
>>> my_sum(*lst1) # unpack the list into separate posit. args
66

>>> lst2 = [44, 55]
>>> my_sum(*lst2)
TypeError: my_sum() missing 1 required positional argument: 'c'
```

---



# Argumentos con palabra clave \*\*kwargs

- Para indicar parámetros con palabras claves se puede usar `**` para empaquetarlos en un diccionario.

```
def my_print_kwargs(**kwargs):
    # Accept variable number of keyword arguments
    """Print all the keyword arguments"""
    for key, value in kwargs.items(): # kwargs is a dict.
        print('%s: %s' %(key, value))
```

```
my_print_kwargs(name='Peter', age=24)
```

```
# use ** to unpack a dict. into individual keyword arguments
dict = {'k1': 'v1', 'k2': 'v2'}
my_print_kwargs(**dict)
# Use ** to unpack dict. into separate keyword args k1=v1, k2=v2
```



# Argumentos variables \*args y \*\*kwargs

---

- Se puede usar ambos \*args y \*\*kwargs en la definición de una función poniendo \*args primero.

```
def my_print_all_args(*args, **kwargs):
# Place *args before **kwargs
    """Print all positional and keyword arguments"""
    for item in args: # args is a tuple
        print(item)
    for key, value in kwargs.items(): #kwargs is dictionary
        print('%s: %s' %(key, value))

my_print_all_args('a', 'b', 'c', name='Peter', age=24)
# Place the positional arguments before the keyword
# arguments during invocation
```



# Valores retornados por una función

---

- Se puede retornar valores múltiples desde una función Python. En realidad retorna una tupla.

```
>>> def my_fun():
    return 1, 'a', 'hello'
```

```
>>> x, y, z = my_fun()
>>> z
'hello'
>>> my_fun()
(1, 'a', 'hello')
```



# Funciones iter() y next()

- La función `iter(iterable)` devuelve un objeto iterator de iterable y con `next(iterator)` para iterar a través de los items.

```
>>> i = iter([11, 22, 33])
>>> next(i)
11
>>> next(i)
22
>>> next(i)
33
>>> next(i) # Raise StopIteration exception if no more item
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>> type(i)
<class 'list_iterator'>
```



# Función range()

- La función range produce una secuencia de enteros.

Formato:

- range(n) produce enteros desde 0 a n-1;
- range(m, n) produce enteros desde m a n-1;
- range(m, n, s) produce enteros desde m a n-1 en paso de s.

```
for num in range(1, 5):  
    print(num)
```

#

Resul

t 1 2

— 3 4 —



# Función range()

```
# Sum from 1 to the given upperbound
upperbound = int(input('Enter the upperbound: '))
sum = 0
for number in range(1, upperbound+1):    # list of 1 to n
    sum += number
print("The sum is: %d" % sum)
# Sum a given list
lst = [9, 8, 4, 5]
sum = 0
for index in range(len(lst)):    # list of 0 to len-1
    sum += lst[index]
print(sum)
# Better alternative of the above
lst = [9, 8, 4, 5]
sum = 0
for item in lst:    # Each item of lst
    sum += item
print(sum)
# Use built-in function
del sum# Need to remove the sum variable before using builtin function sum
print(sum(lst))
```



# Función enumerate()

- 
- Se puede usar la función integrada `enumerate()` para obtener los índices posicionales cuando se recorre a través de una secuencia.

```
# List
>>> for i, v in enumerate(['a', 'b', 'c']): print(i, v)
1 a
2 b
3 c
>>> enumerate(['a', 'b', 'c'])
<enumerate object at 0x7ff0c6b75a50>
```

```
# Tuple
>>> for i, v in enumerate(('d', 'e', 'f')): print(i, v)
1 d
2 e
3 f
```



# Función reversed()

- Se usa para iterar una secuencia en orden inverso.

```
>>> lst = [11, 22, 33]
>>> for item in reversed(lst): print(item, end=' ')
33 22 11
>>> reversed(lst)
<list_reverseiterator object at 0x7fc4707f3828>

>>> str = "hello"
>>> for c in reversed(str): print(c, end='')
olleh
```



# Secuencias múltiples y función zip()

- Para iterar sobre dos o más secuencias de forma concurrente y emparejadas se usa la función zip.

```
>>> lst1 =['a', 'b', 'c']
>>> lst2 =[11, 22, 33]
>>> for i1, i2 in zip(lst1, lst2): print(i1, i2)
a 11
b. 22
c. 33
>>> zip(lst1, lst2) # Return a list of tuples
[('a', 11), ('b', 22), ('c', 33)]
```

```
# zip() for more than 2 sequences
>>> tuple3 =(44, 55)
>>> zip(lst1, lst2, tuple3)
[('a', 11, 44), ('b', 22, 55)]
```



---

# Uso de módulos y paquetes en Python

## Tema 4



# Módulos

- Un módulo Python es un fichero que contiene código Python, incluyendo instrucciones, variables, funciones y clases.
- Debe guardarse con la extensión .py
- El nombre del módulo es el nombre del fichero:  
`<nombre_modulo>.py`
- Típicamente un módulo comienza con una cadena de documentación (triple comilla) que se invoca con  
`<nombre_modulo>.__doc__`



# Instrucción import

---

- Para usar un módulo en un programa se utiliza la instrucción `import`
- Una vez importado, se referencia los atributos del módulo como `<nombre_modulo>.<nombre_atributo>`
- Se usa `import-as` para asignar un nuevo nombre al módulo para evitar conflicto de nombres en el módulo
- Se puede agrupar en el siguiente orden:
  - Librería standard
  - Librerías de terceros
  - Librerías de aplicación local



# Ejemplo módulo e import

- Ejemplo: fichero greet.py

```
"""
greet
-----
This module contains the greeting message 'msg' and
greeting function 'greet()'.

"""

msg = 'Hello'      # Global Variable

def greet(name):   # Function
    print('{}, {}'.format(msg, name))
```



# Ejemplo módulo e import

```
>>> import greet
>>> greet.greet('Peter')                      # <module_name>.<function_name>
Hello, Peter
>>> print(greet.msg)                         # <module_name>.<var_name>
Hello

>>> greet.__doc__                            # module's doc-string
'greet.py:    the greet module with attributes msg and
greet()'
>>> greet.__name__                           # module's name
'greet'

>>> dir(greet) # List all attributes defined in the module
['__builtins__', '__cached__', '__doc__', '__file__',
 '__loader__', '__name__', '__package__', '__spec__',
 'greet', 'msg']
```



# Ejemplo módulo e import

```
>>> help(greet)# Show module's name, functions, data, ...
Help on module greet:
NAME
    greet
DESCRIPTION
    ...doc-string...
FUNCTIONS
    greet(name)
DATA
    msg = 'Hello'
FILE
    /path/to/greet.py

>>> import greet as gr  # Refer. the 'greet' module as 'gr'
>>> gr.greet('Paul')
Hello, Paul
```



# Instrucción from - import

- La sintaxis es:

```
from <module_name> import <attr_name> # import one attribute  
from <module_name> import <attr_name_1>, <attr_name_2>, ... #  
import selected attributes  
from <module_name> import * #import ALL attributes (NOT recomm.)  
from <module_name> import <attr_name> as <name>  
# import attribute as the given name
```

- Con from – import se referencia los atributos importados usando <attr\_name> directamente.

```
>>> from greet import greet, msg as message  
>>> greet('Peter') # Reference without the 'module_name'  
Hello, Peter  
>>> message  
'Hello'  
>>> msg  
NameError: name 'msg' is not defined
```



# Variable de entorno sys.path y PYTHONPATH

---

- El camino de búsqueda de módulos es mantenida por la variable Python path del módulo sys, sys.path
- sys.path es inicializada a partir de la variable de entorno PYTHONPATH. Por defecto incluye el directorio de trabajo en curso.

```
>>> import sys  
>>> sys.path  
[ '', '/usr/lib/python3.5', '/usr/local/lib/python3.5/dist-packages',  
 '/usr/lib/python3.5/dist-packages', ... ]
```



# Packages

---

- Un módulo contiene atributos (variables, funciones y clases). Los módulos relevantes (mantenidos en el mismo directorio) se pueden agrupar en un package.
- Python también soporta sub-packages (en sub-directorios).
- Los packages y sub-packages son una forma de organizar el espacio de nombres en la forma:  
`<pack_name>.<sub_pack_name>.<sub_sub_pack_name>.<module_name>.<attr_name>`



# Plantilla de módulo individual

```
"""
<package_name>.<module_name>
-----
A description to explain functionality of this module.
Class/Function however should not be documented here.
:author: <author-name>
:version: x.y.z (verion.release.modification)
:copyright: .....
:license: .....
"""

import <standard_library_modules>
import <third_party_library_modules>
import <application_modules>
# Define global variables
.....
# Define helper functions
.....
# Define the entry 'main' function
def main():
    """The main function doc-string"""
.....
# Run the main function if
name=='__main__':
    main()
```



# Packages

---

- Para crear un package:
  - Crear un directorio y nombrarlo con el nombre del package
  - Poner los módulos en el directorio
  - Crear un fichero ‘`__init__.py`’ en el directorio para marcar el directorio como un package



# Ejemplo package

```
myapp/                      # This directory is in the 'sys.path'  
|  
+ mypack1/                  # A directory of relevant modules  
|   |  
|   + __init__.py    # Mark as a package called 'mypack1'  
|   + mymod1_1.py    # Reference as 'mypack1.mymod1_1'  
|   + mymod1_2.py    # Reference as 'mypack1.mymod1_2'  
|  
+ mypack2/                  # A directory of relevant modules  
|  
+ __init__.py    # Mark as a package called 'mypack2'  
+ mymod2_1.py    # Reference as 'mypack2.mymod2_1'  
+ mymod2_2.py    # Reference as 'mypack2.mymod2_2'
```



# Ejemplo package

---

- Si 'myapp' está en 'sys.path' se puede importar 'mymod1\_1' como:

```
import mypack1.mymod1_1
# Reference 'attr1_1_1' as
'mypack1.mymod1_1.attr1_1_1'    from    mypack1
import      mymod1_1
# Reference 'attr1_1_1' as 'mymod1_1.attr1_1_1'
```



# Variables locales y globales

---

- Los nombres creados dentro de una función son locales a la función y están disponibles dentro de la función solamente.
- Los nombres creados fuera de las funciones son globales en el módulo y están disponibles dentro de todas las funciones definidas en el módulo.



# Variables locales y globales - ejemplo

---

```
x = 'global'      # x is a global variable for this module

def myfun(arg):  # arg is a local variable for this
function
    y = 'local'  # y is also a local variable
    # Function can access both local and global variables
    print(x)
    print(y)
    print(arg)

myfun( 'abc' )
print(x)
#print(y)  # Locals are not visible outside the function
#print(arg)
```



# Variables función

- A una variable se le puede asignar un valor, una función o un objeto.

```
>>> def square(n): return n * n
>>> square(5)
25
>>> sq = square    # Assign a function to a variable
>>> sq(5)
25
>>> type(square)
<class 'function'>
>>> type(sq)
<class 'function'>
>>> square
<function square at 0x7f0ba7040f28>
>>> sq
<function square at 0x7f0ba7040f28> # same reference square
```



## Variables función

- Se puede asignar una invocación específica de una función a una variable.

```
>>> def square(n): return n * n  
  
>>> sq5 = square(5)    # A specific function invocation  
>>> sq5  
25  
>>> type(sq5)  
<class 'int'>
```



# Funciones anidadas

- Se puede anidar funciones. Definir una función dentro de una función

```
def outer(a):      #Outer  function
    print('outer begins with arg =', a)
    x = 1 # Define a local  variable
    def inner(b): # Define an inner function
        print('inner begins with arg = %s' % b)
        y = 2
        print('a = %s, x = %d, y = %d' % (a, x, y))
        print('inner ends')
    # Call inner function defined earlier
    inner('bbb')
    print('outer ends')
# Call outer funct, which in turn calls the inner function
outer('aaa')
```



# Función lambda

- 
- Las funciones lambda son funciones anónimas o funciones sin nombre. Se usan para definir una función inline. La sintaxis es:

```
lambda arg1, arg2, ...: return-expression
```

```
>>> def f1(a, b, c): return a + b + c # ordinary function
>>> f1(1, 2, 3)
6
>>> type(f1)
<class 'function'>
>>> f2 = lambda a, b, c: a + b + c # Define a Lambda function
>>> f2(1, 2, 3) # Invoke function
6
>>> type(f2)
<class 'function'>
```



# Las funciones son objetos

- Las funciones son objetos, por tanto:
  - Una función se puede asignar a una variable
  - Una función puede ser pasada en una función como argumento
  - Una función puede ser el valor retomado de una función



# Paso de una función como argumento de una función

---

- El nombre de una función es el nombre de una variable que se puede pasar en otra función como argumento.

```
def my_add(x, y):  
    return x + y  
  
def my_sub(x,y):  
    return x - y  
  
def my_apply(func, x, y): # takes a function as first arg  
    return func(x, y) # Invoke the function received  
  
print(my_apply(my_add, 3, 2)) # Output: 5  
print(my_apply(my_sub, 3, 2)) # Output: 1  
  
# We can also pass an anonymous function as argument  
print(my_apply(lambda x, y: x * y, 3, 2)) # Output: 6
```

---



# Nombres, Espacio de nombres (Namespace) y ámbito

---

- Un nombre se aplica a casi todo incluyendo una variable, función, clase/instancia, módulo/package
  - Los nombre definidos dentro de una función son locales a ella. Los nombres definidos fuera de todas las funciones son globales al módulo y son accesibles por todas las funciones dentro del módulo.
  - Un espacio de nombres (namespace) es una colección de nombres.
  - El ámbito se refiere a la porción del programa a partir de la cual un nombre se puede acceder sin prefijo.
-



# Cada módulo tiene un Espacio de nombres Global

---

- Un módulo es un fichero que contiene atributos (variables, funciones y clases) y tiene su propio espacio de nombres globales.
  - Por ello no se puede definir dos funciones o clases con el mismo nombre dentro de un módulo, pero sí en diferentes módulos.
- Cuando se ejecuta el Shell interactivo, Python crea un módulo llamado `__main__`, con su namespace global asociado.



# Cada módulo tiene un Espacio de nombres Global

---

- Cuando se importa un módulo con ‘import <module\_name>’:
  - En caso de Shell interactivo, se añade <module\_name> al namespace de `__main__`
  - Dentro de otro módulo se añade el nombre al namespace del módulo donde se ha importado.
- Si se importa un atributo con ‘from <module\_name> import <attr\_name>’ el <attr\_name> se añade al namespace de `__main__`, y se puede acceder al <attr\_name> directamente.



# Funciones `globals()`, `locals()` y `dir()`

---

- Se puede listar los nombres del ámbito en curso con las funciones integradas:
  - `globals()`: devuelve un diccionario con las variables globales en curso
  - `locals()`: devuelve un diccionario con las variables locales.
  - `dir()`: devuelve una lista de los nombres locales, que es equivalente a `locals().keys()`



# Modificación de variables globales dentro de una función

---

- Para modificar una variable global dentro de una función se usa la instrucción `global`.

```
x = 'global'          # Global file-scope

def myfun():
    global x      # Declare x global to modify global variable
    x =
        'change'
    print(x)
myfun()
print(x)            # Global changes
```



# Funciones - terminología

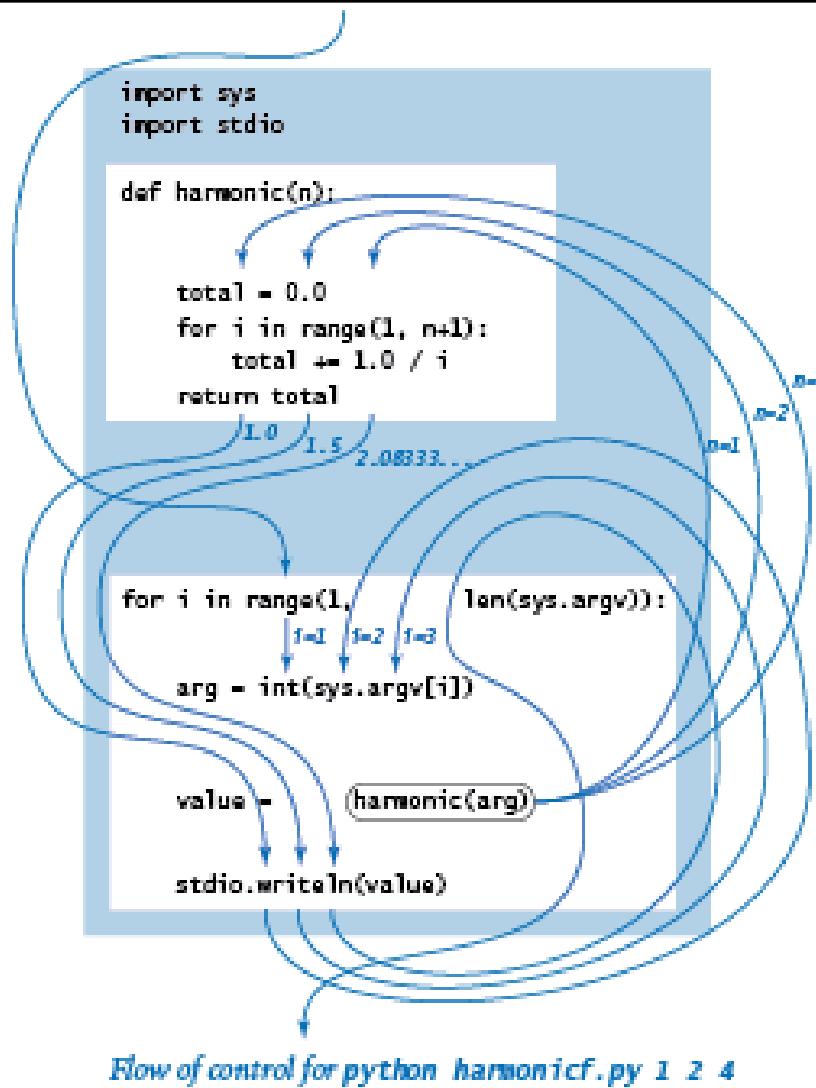
---

<i>concept</i>	<i>Python construct</i>	<i>description</i>
<i>function</i>	function	mapping
<i>input value</i>	argument	input to function
<i>output value</i>	return value	output of function
<i>formula</i>	function body	function definition
<i>independent variable</i>	parameter variable	symbolic placeholder for input value



# Funciones – control de flujo

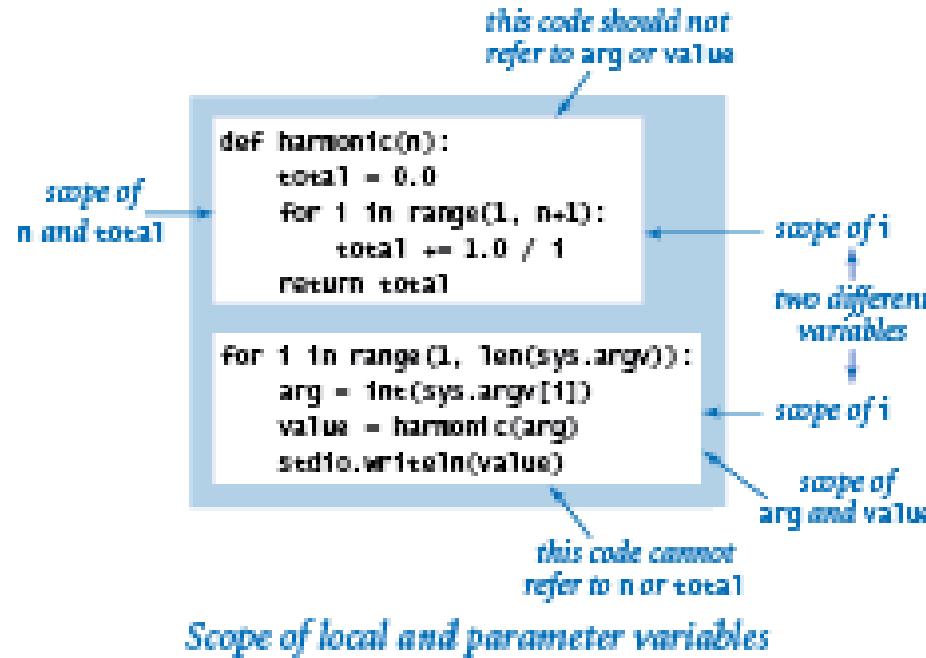
- import
- def
- return





# Funciones – alcance

- Las variables son locales en el bloque donde se definen





# Funciones – código típico

*primality test*

```
def isPrime(n):
    if n < 2: return False
    i = 2
    while i*i <= n:
        if n % i == 0: return False
        i += 1
    return True
```

*hypotenuse of a right triangle*

```
def hypot(a, b)
    return math.sqrt(a*a + b*b)
```

*generalized harmonic number*

```
def harmonic(n, r=1):
    total = 0.0
    for i in range(1, n+1):
        total += 1.0 / (i ** r)
    return total
```

*draw a triangle*

```
def drawTriangle(x0, y0, x1, y1, x2, y2):
    stddraw.line(x0, y0, x1, y1)
    stddraw.line(x1, y1, x2, y2)
    stddraw.line(x2, y2, x0, y0)
```

*Typical code for implementing functions*



# Funciones - Paso de argumentos

---

- Los argumentos de tipo integer, float, boolean, o string por valor. El resto de objetos se pasan por referencia.



# Funciones – código típico con arrays

<i>mean of an array</i>	<pre>def mean(a):     total = 0.0     for v in a:         total += v     return total / len(a)</pre>
<i>dot product of two vectors of the same length</i>	<pre>def dot(a, b):     total = 0     for i in range(len(a)):         total += a[i] * b[i]     return total</pre>
<i>exchange two elements in an array</i>	<pre>def exchange(a, i, j):     temp = a[i]     a[i] = a[j]     a[j] = temp</pre>
<i>write a one-dimensional array (and its length)</i>	<pre>def write1D(a):     stdio.writeln(len(a))     for v in a:         stdio.writeln(v)</pre>
<i>read a two-dimensional array of floats (with dimensions)</i>	<pre>def readFloat2D():     m = stdio.readInt()     n = stdio.readInt()     a = stdarray.create2D(m, n, 0.0)     for i in range(m):         for j in range(n):             a[i][j] = stdio.readFloat()     return a</pre>

*Typical code for implementing functions with arrays*



# Funciones - recursión

- Técnica de programación utilizada en muchas aplicaciones. Capacidad de invocar una función desde la misma función.

```
import sys
# Return n!
def factorial(n):
    if n==1:
        return 1
    return n * factorial(n-1)
def main():
    n=int(sys.argv[1])
    fact=factorial(n)
    print(fact)
if __name__=='__main__':
    main()
# python factorial.py 3
# 6
```

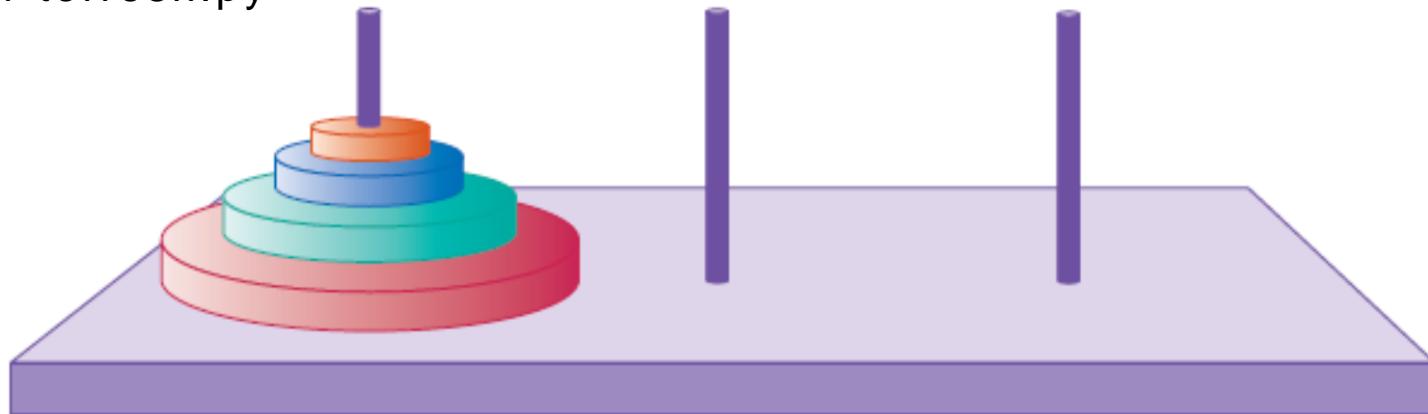


# Funciones - recursión

```
# Imprime los movimientos para resolver las torres de hanoi
# parametros: numero discos, torre partida, torre final, torre auxiliar
def mover(discos, detorre, atorre, auxtorre) :
    if discos >= 1 :
        mover(discos - 1, detorre, auxtorre, atorre)
        print("Mover disco ", discos, " de ", detorre, " a ", atorre)
        mover(discos - 1, auxtorre, atorre, detorre)

def main() :
    mover(5, "A", "C", "B")

if __name__=='__main__':
    main()
# python torresh.py
```





# Funciones como objetos

- En Python cada elemento es un objeto, incluyendo funciones.

```
# Fichero integ.py
# Calcula la integral de Riemann de una function f
def integrate(f, a, b, n=1000):
    total = 0.0
    dt = 1.0 * (b - a) / n
    for i in range(n):
        total += dt * f(a +(i + 0.5) * dt)
    return total
```



# Funciones como objetos

---

```
# Fichero intdrive.py
import funarg as fa
def square(x):
    return x*x

def main():
    print(fa.integrate(square,0, 10)

if __name__== '__main__':
    main()
```

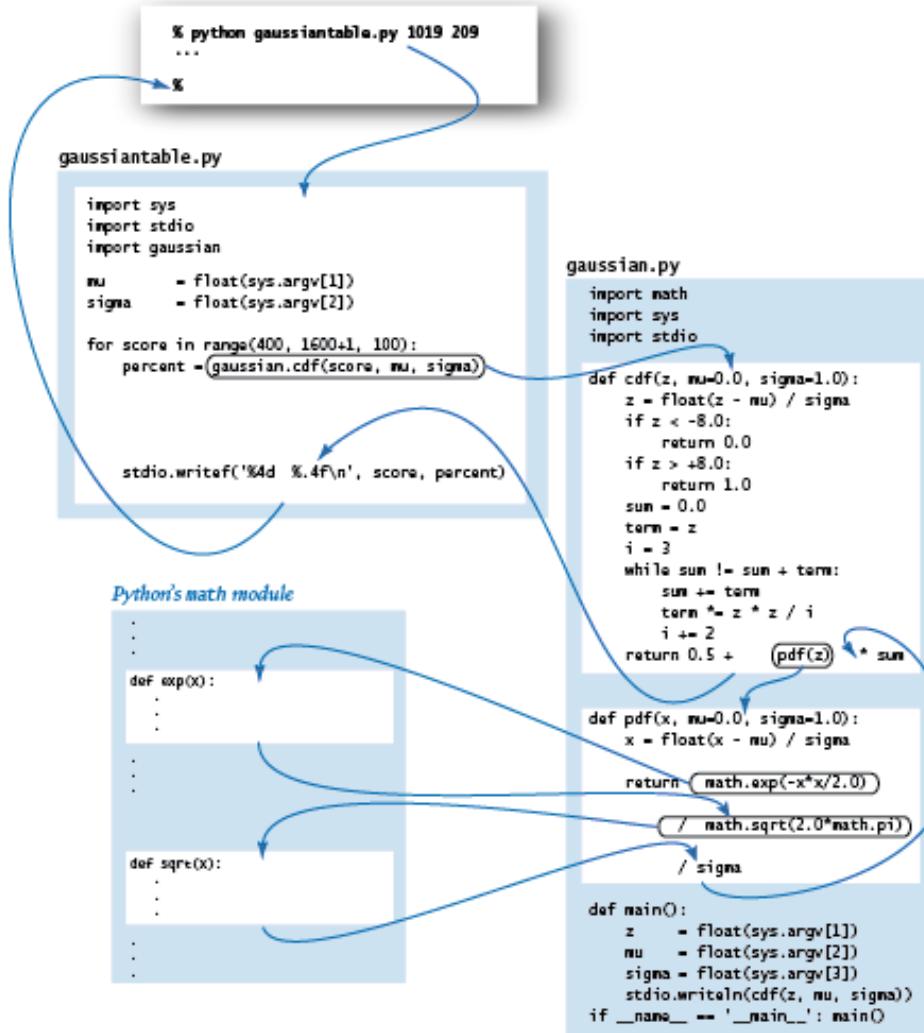


# Módulos

- Un *módulo* contiene funciones que están disponibles para su uso en otros programas.
- Un *cliente* es un programa que hace uso de una función en un módulo.
- Pasos:
  - En el cliente: import el módulo.
  - En el cliente: hacer llamada a la función.
  - En el módulo: colocar una prueba de cliente main().
  - En el módulo: eliminar código global.  
Usar if \_\_\_\_\_name\_\_\_\_\_='\_\_main\_\_':  
main()
  - Hacer accesible el módulo para el cliente.



# Módulos



Flow of control in a modular program



# Programación modular

- Implementaciones.
- Clientes.
- Application programming interfaces (APIs).

<i>function call</i>	<i>description</i>
<code>gaussian.pdf(x, mu, sigma)</code>	<i>Gaussian probability density function</i> $\phi(x, \mu, \sigma)$
<code>gaussian.cdf(z, mu, sigma)</code>	<i>Gaussian cumulative distribution function</i> $\Phi(x, \mu, \sigma)$

*Note: The default value for mu is 0.0 and for sigma is 1.0.*

*API for our gaussian module*

- Funciones privadas:
  - Funciones que solo se usan en los módulos y que no se ofrecen a los clientes. Por convención se usa un guión bajo como primer carácter del nombre de la función.



# Programación modular

---

- **Librerías:**
  - Colección de módulos relacionados. Ejemplo: NumPy, Pygame, Matplotlib, SciPy, SymPy, Ipython.
- **Documentación.**

```
>>> import stddraw  
>>> help stddraw
```



# Assertion and Exception Handling - assert

---

- Instrucción assert. Se usa para probar una aserción.
  - Sintaxis:

```
assert test, error-message
```

```
>>> x = 0
>>> assert x == 0, 'x is not zero?!' # Assertion true, no
output

>>> x = 1
# Assertion false, raise AssertionError with the message
>>> assert x == 0, 'x is not zero?'
.....
AssertionError: x is not zero?!
```



# Assertion and Exception Handling - Exceptions

---

- Los errores detectados durante la ejecución se llaman excepciones. Cuando se produce el programa termina abruptamente.

```
>>> 1/0          # Divide by 0
.....
ZeroDivisionError: division by zero
>>> zzz          # Variable not defined
.....
NameError: name 'zzz' is not defined
>>> '1' + 1      # Cannot concatenate string and int
.....
TypeError: Can't convert 'int' object to str implicitly
```



# Assertion and Exception Handling - Exceptions

---

```
>>> lst = [0, 1, 2]
>>> lst[3]      # Index out of range
.....
IndexError: list index out of range
>>> lst.index(8) # Item is not in the list
.....
ValueError: 8 is not in list

>>> int('abc')    # Cannot parse this string into int
.....
ValueError: invalid literal for int() with base 10: 'abc'

>>> tup = (1, 2, 3)
>>> tup[0] = 11   # Tuple is immutable
.....
TypeError: 'tuple' object does not support item assignment
```



# Assertion and Exception Handling – try-except-else-finally

---

- Sintaxis:

```
try:  
    statements  
except exception- 1:                      # Catch one exception  
    statements  
except (exception- 2, exception- 3): # Catch multiple except.  
    statements  
except exception- 4 as var_name: # Retrieve the excep. inst  
    statements  
except:          # For (other) exceptions  
    statements  
else:  
    statements      # Run if no exception raised  
finally:  
    statements      # Always run regardless of whether  
exception raised
```

---



# Assertion and Exception Handling – try-except-else-finally

---

- Ejemplo 1: Gestión de índice fuera de rango en acceso a lista: ejem1\_excep.py
- Ejemplo 2: Validación de entrada.

```
>>> while True:  
    try:  
        x = int(input('Enter an integer: '))  
        break  
    except ValueError:  
        print('Wrong input! Try again...')      # Repeat
```

```
Enter an integer: abc  
Wrong input! Try again..  
Enter an integer: 11.22  
Wrong input! Try again..  
Enter an integer: 123
```



# Instrucción with-as y gestores de contexto

- La sintaxis de with-as es:

```
with... as ...:  
    statements
```

```
# More than one items
```

```
with... as ..., ... as ..., ...:  
    statements
```

- Ejemplos:

```
# automatically close the file at the end of with  
with open('test.log', 'r') as infile:  
    for line in infile:  
        print(line)
```



# Instrucción with-as y gestores de contexto

- **Ejemplos:**

```
# automatically close the file at the end of with
with open('test.log', 'r') as infile:
    for line in infile:
        print(line)
```

```
# Copy a file
with open('in.txt', 'r') as infile, open('out.txt', 'w') as outfile:
    for line in infile:
        outfile.write(line)
```



# Módulos de librería standard Python de uso común

---

- Python dispone de un conjunto de librerías standard.
- Para usarlas se usa ‘import <nombre\_modulo>’ o ‘from <nombre\_modulo> import <nombre\_atributo>’ para importar la librería completa o el atributo seleccionado.

```
>>> import math    # import an external module
>>> dir(math)    # List all attributes
['e', 'pi', 'sin', 'cos', 'tan', 'tan2', ...]
>>> help(math)
...
>>> help(math.atan2)
...
```



# Módulos de librería standard Python de uso común

---

```
>>> math.atan2(3, 0)
1.5707963267948966
>>> math.sin(math.pi / 2)
1.0
>>> math.cos(math.pi / 2)
6.123233995736766e-17

>>> from math import pi
>>> pi
3.141592653589793
```



# Módulos math y cmath

---

- El módulo `math` proporciona acceso las funciones definidas por el lenguaje C. Los más comunes son:
  - Constantes: `pi`, `e`.
  - Potencia y exponente: `pow(x,y)`, `sqrt(x)`, `exp(x)`, `log(x)`, `log2(x)`, `log10(x)`
  - Conversión float a int: `ceil(x)`, `floor(x)`, `trunc(x)`
  - Operaciones float: `fabs()`, `fmod()`
  - `hypot(x,y)` ( $=\sqrt{x^*x + y^*y}$ )
  - Conversión entre grados y radianes: `degrees(x)`, `radians(x)`
  - Funciones trigonométricas: `sin(x)`, `cos(x)`, `tan(x)`, `acos(x)`, `asin(x)`, `atan(x)`, `atan2(x,y)`
  - Funciones hiperbólicas: `sinh(x)`, `cosh(x)`, `tanh(x)`, `asinh(x)`, `acosh(x)`, `atanh(x)`



# Módulo statistics

- El módulo `statistics` calcula las propiedades estadísticas básicas.

```
>>> import statistics
>>> dir(statistics)
['mean', 'median', 'median_grouped', 'median_high',
'median_low', 'mode', 'pstdev', 'pvariance',
'stdev', 'variance', ...]
>>> help(statistics)
.....
>>> help(statistics.pstdev)
.....
>>> data = [5, 7, 8, 3, 5, 6, 1, 3]
>>> statistics.mean(data)
4.75
```



# Módulo statistics

---

```
>>> statistics.median(data)
5.0
>>> statistics.stdev(data)
2.3145502494313788
>>> statistics.variance(data)
5.357142857142857
>>> statistics.mode(data)
statistics.StatisticsError: no unique mode; found 2 equally
common values
```



# Módulo random

- El módulo random se usa para generar números pseudo random.

```
>>> import random  
>>> dir(random)  
[...]  
>>> help(random)  
[...]  
>>> help(random.random)  
[...]  
  
>>> random.random()          # float in [0,1)  
0.7259532743815786  
>>> random.random()  
0.9282534690123855
```



# Módulo random

- 
- El módulo random se usa para generar números pseudo random.

```
>>> random.randint(1, 6) # int in [1,6]
3
>>> random.randrange(6)    # From range(6), i.e., 0 to 5
0
>>> random.choice(['apple', 'orange', 'banana'])
'apple'
```



# Módulo sys

- 
- El módulo `sys` (de `system`) proporciona parámetros y funciones específicos de sistema. Los más usados:
    - `sys.exit([exit-status=0])`: salir del programa.
    - `sys.path`: Lista de rutas de búsqueda. Initializado con la variable de entorno `PYTHONPATH`.
    - `sys.stdin`, `sys.stdout`, `sys.stderr`: entrada, salida y error estándard.
    - `sys.argv`: Lista de argumentos en la línea de comandos



# Módulo sys

- Script `test_argv.py`

```
import sys
print(sys.argv)          # Print command-line argument list
print(len(sys.argv))    # Print length of list
```

- Ejecución del script

```
$ python test_argv.py
['test_argv.py']
1
```

```
$ python test_argv.py hello 1 2 3 apple orange
['test_argv.py', 'hello', '1', '2', '3', 'apple', 'orange']
# list of strings
7
```



# Módulo os

- 
- El módulo os proporciona una interfaz con el sistema operativo. Los atributos más usados son:
    - `os.mkdir(path, mode=0777)`: Crea un directorio
    - `os.makedirs(path, mode=0777)`: Similar a mkdir
    - `os.getcwd()`: devuelve el directorio en curso
    - `os.chdir(path)`: Cambia el directorio en curso
    - `os.system(command)`: ejecuta un comando shell.
    - `os.getenv(varname, value=None)`: devuelve la variable de entorno si existe
    - `os.putenv(varname, value)`: asigna la variable de entorno al valor
    - `os.unsetenv(varname)`: elimina la variable de entorno



# Módulo os

- **Ejemplo:**

```
>>> import os
>>> dir(os)          # List all attributes
.....
>>> help(os)         # Show man page
.....
>>> help(os.getcwd)  # Show man page for specific function
.....
>>> os.getcwd()       # Get current working directory
...current working directory...
>>> os.listdir('.')   # List contents of the current direct
...contents of current directory...
>>> os.chdir('test-python')    # Change directory
>>> exec(open('hello.py').read()) # Run a Python script
>>> os.system('ls -l')        # Run shell command
```



# Módulo os

- Ejemplo:

```
>>> import os  
>>> os.name                      # Name of OS  
'posix'  
>>> os.makedirs(dir)            # Create sub-directory  
>>> os.remove(file)              # Remove file  
>>> os.rename(oldFile, newFile)  # Rename file  
>>> os.listdir('.')             # Return a list of entries in the given directory  
>>> for f in sorted(os.listdir('.')):  
    print(f)
```



## Módulo date

- Proporciona clases para la manipulación de fechas y tiempos

```
>>> import datetime
>>> dir(datetime)
['MAXYEAR', 'MINYEAR', 'date', 'datetime', 'datetime_CAPI',
'time', 'timedelta', 'timezone', 'tzinfo', ...]
>>> dir(datetime.date)
['today', ...]

>>> from datetime import date
>>> today = date.today()
>>> today
datetime.date(2016, 6, 17)
```



# Módulo date

- Proporciona clases para la manipulación de fechas y tiempos

```
>>> import datetime  
>>> aday = date(2016, 5, 1) # Construct a datetime.date i  
>>> aday  
datetime.date(2016, 5, 1)  
>>> diff = today - aday # Find the diff between 2 dates  
>>> diff  
datetime.timedelta(47)  
>>> dir(datetime.timedelta)  
['days', 'max', 'microseconds', 'min', 'resolution',  
'seconds', 'total_seconds', ...]  
>>> diff.days  
47
```



## Módulo time

- Se puede usar para medir el tiempo de ejecución de un script

```
import time  
start = time.time()  
  
“codigo que se desea medir el tiempo aqui ”  
  
end = time.time()  
print(end - start)
```



# Sympy

- 
- Sympy es una librería que permite hacer operaciones simbólicas en lugar de con valores numéricos
  - [Portal Sympy](#)
  - [SymPy Tutorial](#)



# Scipy

---

- Librería de funciones matemáticas para cálculo numérico tales como integración y optimización
- [Portal](#)
- [Tutorial](#)



---

# Programación orientada a objetos en Python

## Tema 5



# Programación orientada a objetos (OOP) en Python

---

- Una clase es una plantilla de entidades del mismo tipo. Una instancia es una realización particular de una clase. Python soporta instancias de clases y objetos.
  - Un objeto contiene atributos: atributos de datos (o variables) y comportamiento (llamados métodos). Para acceder un atributo se usa el operador punto, ejemplo: `nombre_instancia.nombre_atributo`
  - Para crear una instancia de una clase se invoca el constructor: `nombre_instancia = nombre_clase(*args)`
-



# Objetos de clase vs Objetos de instancia

---

- Los objetos de clase sirven como factorías para generar objetos de instancia. Los objetos instanciados son objetos reales creados por una aplicación. Tienen su propio espacio de nombres.
- La instrucción `class` crea un objeto de clase con el nombre de clase. Dentro de la definición de la clase se puede crear variables de clase y métodos mediante `def`s que serán compartidos por todas las instancias



# Sintaxis de la definición de clase

- La sintaxis es:

```
class class_name(superclass1, ...):  
    """Class doc-string"""  
    class_var1 = value1 # Class variables  
  
    .....  
    def __init__(self, arg1, ...):  
        """Constructor"""  
        self.instance_var1 = arg1 # inst var by assignment  
  
    .....  
    def __str__(self):  
        """For printf() and str()"""  
  
    .....  
    def __repr__(self):  
        """For repr() and interactive prompt"""  
  
    .....  
    def method_name(self, *args, **kwargs):  
        """Method doc-string"""  
    .....
```



# Contructor: Self

- El primer parámetro de un constructor debe ser self
- Cuando se invoca el constructor para crear un nuevo objeto, el parámetro self se asigna al objeto que está siendo inicializado

```
def __init__(self):  
    self._itemCount = 0  
    self._totalPrice = 0
```

```
register = CashRegister()
```

Referencia al  
objeto inicializado

Cuando el constructor termina this es  
la referencia al objeto creado



# Ejemplo

- `circle.py`:

```
from math import pi
```

```
class Circle:  
    """A Circle instance models a circle with a radius"""  
    def __init__(self, radius=1.0):  
        """Constructor with default radius of 1.0"""  
        self.radius = radius# Create an instance variable  
    def __str__(self):  
        """Return string, invoked by print() and str()"""  
        return 'circle with radius of %.2f' % self.radius  
    def __repr__(self):  
        """Return string used to re-create this instance"""  
        return 'Circle(radius=%f)' % self.radius  
    def get_area(self):  
        """Return the area of this Circle instance"""  
        return self.radius * self.radius * pi
```



# Ejemplo

- `circle.py` (cont.):

```
# if run under Python interpreter, __name__ is '__main__'
# If imported into another module, __name__ is
'circle'
if __name__ == '__main__': # Construct an instance
    print(c1)             # Invoke __str__()
    print(c1.get_area())
    c2 = Circle()          # Default radius
    print(c2)
    print(c2.get_area())   # Invoke member method
    c2.color = 'red'        # Create new attribute via assignment
    print(c2.color)
    #print(c1.color)       # Error - c1 has no attribute color
    # Test doc-strings
    print(__doc__)          # This module
    print(Circle.__doc__)    # Circle class
    print(Circle.get_area.__doc__) # get_area() method
    print(isinstance(c1, Circle)) # True
```



# Construcción de clase

- Para construir una instancia de una clase se realiza a través del constructor `nombre_clase(...)`

```
c1 = Circle(1.2)
c2 = Circle()      # radius default
```

- Python crea un objeto `Circle`, luego invoca el método `__init__(self, radius)` con `self` asignado a la nueva instancia
  - `__init__()` es un inicializador para crear variables de instancia
  - `__init__()` nunca devuelve un valor
  - `__init__()` es opcional y se puede omitir si no hay variables de instancia



# Clase Point y sobrecarga de operadores

- point.py modela un punto 2D con coordenadas x e y.  
**Se sobrecarga los operadores + y \*:**

```
""" point.py: point module defines the Point class"""
class Point:
    """A Point models a 2D point x and y coordinates"""
    def __init__(self, x=0,y=0):
        """Constructor x and y with default of (0,0)"""
        self.x = x
        self.y = y
    def __str__(self):
        """Return a descriptive string for this instance"""
        return '(% .2f, % .2f)' %(self.x, self.y)
    def __add__(self, right):
        """Override the '+' operator"""
        p = Point(self.x + right.x, self.y + right.y)
        return p
```



# Clase Point y sobrecarga de operadores

```
def __mul__(self, factor):
    """Override the '*' operator"""
    self.x *= factor
    self.y *= factor
    return self

# Test
if __name__ == '__main__':
    p1 = Point()
    print(p1)          # (0.00, 0.00)
    p1.x = 5
    p1.y = 6
    print(p1)          # (5.00, 6.00)
    p2 = Point(3, 4)
    print(p2)          # (3.00, 4.00)
    print(p1 + p2)    # (8.00, 10.00) Same as p1.add(p2)
    print(p1)          # (5.00, 6.00) No change
    print(p2 * 3)     # (9.00, 12.00) Same as p1.__mul__(p2)
    print(p2)          # (9.00, 12.00) Changed
```



# Herencia

- cylinder.py un cilindro se puede derivar de un circulo

```
"""cylinder.py: which defines the Cylinder class"""
from circle import Circle # Using the Circle class
class Cylinder(Circle):
    """The Cylinder class is a subclass of Circle"""
    def __init__(self, radius = 1.0, height = 1.0):
        """Constructor"""
        super().__init__(radius) # Invoke superclass
        self.height = height
    def __str__(self):
        """Self Description for print()"""
        return 'Cylinder(radius=%2f,height=%2f)' % (self.radius,
                                                       self.height)
    def get_volume(self):
        """Return the volume of the cylinder"""
        return self.get_area() * self.height
```



# Herencia

- cylinder.py (cont.)

```
if __name__ == '__main__':
    cy1 = Cylinder(1.1, 2.2)
    print(cy1)                      # inherited superclass' method
    print(cy1.get_area())            # Invoke its method
    print(cy1.get_volume())          # Default radius and height
    cy2 = Cylinder()
    print(cy2)
    print(cy2.get_area())
    print(cy2.get_volume())
    print(dir(cy1))
    print(Cylinder.get_area)
    print(Circle.get_area)
    print(Circle(3.3))              # Output: circle with radius of 3.30
    print(isinstance(Cylinder, Circle)) # True
    print(isinstance(Circle, Cylinder)) # False
    print(isinstance(cy1, Cylinder))  # True
```



# Métodos mágicos

Magic Method	Invoked Via	Invocation Syntax
<code>__lt__(self, right)</code> <code>__gt__(self, right)</code> <code>__le__(self, right)</code> <code>__ge__(self, right)</code> <code>__eq__(self, right)</code> <code>__ne__(self, right)</code>	<b>Comparison Operators</b>	<code>self &lt; right</code> <code>self &gt; right</code> <code>self &lt;= right</code> <code>self &gt;= right</code> <code>self == right</code> <code>self != right</code>
<code>__add__(self, right)</code> <code>__sub__(self, right)</code> <code>__mul__(self, right)</code> <code>__truediv__(self, right)</code> <code>__floordiv__(self, right)</code> <code>__mod__(self, right)</code> <code>__pow__(self, right)</code>	<b>Arithmetic Operators</b>	<code>self + right</code> <code>self - right</code> <code>self * right</code> <code>self / right</code> <code>self // right</code> <code>self % right</code> <code>self ** right</code>



# Métodos mágicos

Magic Method	Invoked Via	Invocation Syntax
<code>__and__(self, right)</code> <code>__or__(self, right)</code> <code>__xor__(self, right)</code> <code>__invert__(self)</code> <code>__lshift__(self, n)</code> <code>__rshift__(self, n)</code>	<b>Bitwise Operators</b>	<code>self &amp; right</code> <code>self   right</code> <code>self ^ right</code> <code>~self</code> <code>self &lt;&lt; n</code> <code>self &gt;&gt; n</code>
<code>__str__(self)</code> <code>__repr__(self)</code> <code>__sizeof__(self)</code>	<b>Function call</b>	<code>str(self), print(self)</code> <code>repr(self)</code> <code>sizeof(se lf)</code>
<code>__len__(self)</code> <code>__contains__(self, item)</code> <code>__iter__(self)</code> <code>__next__(self)</code> <code>__getitem__(self, key)</code> <code>__setitem__(self, key, value)</code> <code>__delitem__(self, key)</code>	<b>Sequence Operators &amp; Functions</b>	<code>len(self)</code> <code>item in self</code> <code>iter(self)</code> <code>next(self)</code> <code>self[key]</code> <code>self[key] = value</code> <code>del self[key]</code>



# Métodos mágicos

Magic Method	Invoked Via	Invocation Syntax
<code>__int__(self)</code> <code>__float__(self)</code> <code>__bool__(self)</code> <code>__oct__(self)</code> <code>__hex__(self)</code>	<b>Type Conversion Function call</b>	<code>int(self)</code> <code>float(se</code> <code>lf)</code> <code>bool(se</code> <code>lf)</code> <code>oct(self)</code> <code>hex(self)</code>
<code>__init__(self, *args)</code> <code>__new__(cls, *args)</code>	<b>Constructor</b>	<code>x = ClassName(*args)</code>
<code>__del__(self)</code>	<b>Operator del</b>	<code>del x</code>
<code>__index__(self)</code>	<b>Convert this object to an index</b>	<code>x[self]</code>
<code>__radd__(self, left)</code> <code>__rsub__(self, left)</code> ...	<b>RHS (Reflected) addition, subtraction, etc.</b>	<code>left + self</code> <code>left - self</code> ...
<code>__iadd__(self, right)</code> <code>__isub__(self, right)</code> ...	<b>In-place addition, subtraction, etc</b>	<code>self += right</code> <code>self -= right</code> ...



# Métodos mágicos

Magic Method	Invoked Via	Invocation Syntax
<code>__pos__(self)</code> <code>__neg__(self)</code>	<b>Unary Positive and Negative operators</b>	<code>+self</code> <code>-self</code>
<code>__round__(self)</code> <code>__floor__(self)</code> <code>__ceil__(self)</code> <code>__trunc__(self)</code>	<b>Function Call</b>	<code>round(self)</code> <code>floor(self)</code> <code>ceil(self)</code> <code>trunc(se lf)</code>
<code>__getattr__(self, name)</code> <code>__setattr__(self, name, value)</code> <code>__delattr__(self, name)</code>	<b>Object's attributes</b>	<code>self.name</code> <code>self.name = value</code> <code>del self.name</code>
<code>__call__(self, *args, **kwargs)</code>	<b>Callable Object</b>	<code>obj(*args, **kwargs);</code>
<code>__enter__(self), __exit__(self)</code>	Context Manager with-statement	



# Números random

- Módulo stdrandom.py

<i>function call</i>	<i>description</i>
<code>uniformInt(lo, hi)</code>	<i>uniformly random integer in the range [lo, hi)</i>
<code>uniformFloat(lo, hi)</code>	<i>uniformly random float in the range [lo, hi)</i>
<code>bernoulli(p)</code>	<i>True with probability p (p defaults to 0.5)</i>
<code>binomial(n, p)</code>	<i>number of heads in n coin flips, each of which is heads with probability p (p defaults to 0.5)</i>
<code>gaussian(mu, sigma)</code>	<i>normal, mean mu, standard deviation sigma (mu defaults to 0.0, sigma defaults to 1.0)</i>
<code>discrete(a)</code>	<i>i with probability proportional to a[i]</i>
<code>shuffle(a)</code>	<i>randomly shuffle the array a []</i>

*API for our stdrandom module*



# Procesado de arrays

- Módulo stdarray.py

<i>function call</i>	<i>description</i>
<code>create1D(n, val)</code>	<i>array of length n, each element initialized to val</i>
<code>create2D(m, n, val)</code>	<i>m-by-n array, each element initialized to val</i>
<code>readInt1D()</code>	<i>array of integers, read from standard input</i>
<code>readInt2D()</code>	<i>two-dimensional array of integers, read from standard input</i>
<code>readFloat1D()</code>	<i>array of floats, read from standard input</i>
<code>readFloat2D()</code>	<i>two-dimensional array of floats, read from standard input</i>
<code>readBool1D()</code>	<i>array of booleans, read from standard input</i>
<code>readBool2D()</code>	<i>two-dimensional array of booleans, read from standard input</i>
<code>write1D(a)</code>	<i>write array a[] to standard output</i>
<code>write2D(a)</code>	<i>write two-dimensional array a[] to standard output</i>

*Note 1: 1D format is an integer n followed by n elements.*

*2D format is two integers m and n followed by m × n elements in row-major order.*

*Note 2: Booleans are written as 0 and 1 instead of False and True.*

*API for our stdarray module*



# Estadística

- Módulo stdstats.py

<i>function call</i>	<i>description</i>
<code>mean(a)</code>	<i>average of the values in the numeric array a[]</i>
<code>var(a)</code>	<i>sample variance of the values in the numeric array a[]</i>
<code>stddev(a)</code>	<i>sample standard deviation of the values in the numeric array a[]</i>
<code>median(a)</code>	<i>median of the values in the numeric array a[]</i>
<code>plotPoints(a)</code>	<i>point plot of the values in the numeric array a[]</i>
<code>plotLines(a)</code>	<i>line plot of the values in the numeric array a[]</i>
<code>plotBars(a)</code>	<i>bar plot of the values in the numeric array a[]</i>
	<i>API for our stdstats module</i>



# Beneficios de la programación modular

---

- Programas de tamaño razonable.
- Depuración.
- Reusabilidad de código.
- Mantenimiento.



# Programación orientada a objetos - Métodos

---

- Un método es una función asociada a un objeto específico.
- Se invoca utilizando el nombre del objeto seguido del operador punto (.) seguido por el nombre del método y los argumentos del mismo.

	<i>method</i>	<i>function</i>
<i>sample call</i>	<code>x.bit_length()</code>	<code>stdio.writeln(bits)</code>
<i>typically invoked with</i>	<i>variable name</i>	<i>module name</i>
<i>parameters</i>	<i>object reference and argument(s)</i>	<i>argument(s)</i>
<i>primary purpose</i>	<i>manipulate object value</i>	<i>compute return value</i>

*Methods versus functions*

---



# Programación orientada a objetos – Métodos de la clase str

<i>translate from DNA to mRNA (replace 'T' with 'U')</i>	<pre>def translate(dna):     dna = dna.upper()     rna = dna.replace('T', 'U')     return rna</pre>
<i>is the string s a palindrome?</i>	<pre>def isPalindrome(s):     n = len(s)     for i in range(n // 2):         if s[i] != s[n-1-i]:             return False     return True</pre>
<i>extract file name and extension from a command-line argument</i>	<pre>s = sys.argv[1] dot = s.find('.') base = s[:dot] extension = s[dot+1:]</pre>
<i>write all lines on standard input that contain a string specified as a command-line argument</i>	<pre>query = sys.argv[1] while stdio.hasNextLine():     s = stdio.readLine()     if query in s:         stdio.writeln(s)</pre>
<i>is an array of strings in ascending order?</i>	<pre>def isSorted(a):     for i in range(1, len(a)):         if a[i] &lt; a[i-1]:             return False     return True</pre>

*Typical string-processing code*



# Tipo de dato definido por el usuario

- Se define un tipo Charge para partículas cargadas.
- Se usa la ley de Coulomb para el cálculo del potencial de un punto debido a una carga  $V=kq/r$ , donde q es el valor de la carga, r es la distancia del punto a la carga y  $k=8.99 \times 10^9 \text{ N m}^2/\text{C}^2$ .

## Constructor

<i>operation</i>	<i>description</i>
<code>Charge(x0, y0, q0)</code>	<i>a new charge centered at (x0, y0) with charge value q0</i>
<code>c.potentialAt(x, y)</code>	<i>electric potential of charge c at point (x, y)</i>
<code>str(c)</code>	<i>'q0 at (x0, y0)' (string representation of charge c)</i>

*API for our user-defined Charge data type*



# Convenciones sobre ficheros

---

- El código que define el tipo de dato definido por el usuario Charge se coloca en un fichero del mismo nombre (sin mayúscula) charge.py
- Un programa cliente que usa el tipo de dato Charge se pone en el cabecero del programa:

```
from charge import Charge
```



# Creación de objetos, llamada de métodos y representación de String

```
#-----
# chargeclient.py
#-----
import sys
import stdio
from charge import Charge
#Acepta floats x e y como argumentos en la línea de comandos. Crea dos objetos
#Charge con posición y carga. Imprime el potencial en (x, y) en la salida estandard x =
float(sys.argv[1])
y = float(sys.argv[2])
c1 = Charge(.51, .63, 21.3)
c2 = Charge(.13, .94, 81.9)
v1 = c1.potentialAt(x, y)
v2 = c2.potentialAt(x, y)
stdio.writef('potential at (%.2f, %.2f) due to\n', x, y)
stdio.writeln(    ' +str(c1) +' and')
'           ' + str(c2))
stdio.writeln('is %.2e\n', v1+v2)
#-----
# python chargeclient.py .2 .5
# potential at (0.20, 0.50) due to #
#     21.3 at (0.51, 0.63) and
#     81.9 at (0.13, 0.94)
# is 2.22e+12from charge import Charge
```



# Elementos básicos de un tipo de dato

- API

<i>operation</i>	<i>description</i>
Charge( $x_0$ , $y_0$ , $q_0$ )	<i>a new charge centered at (<math>x_0, y_0</math>) with charge value <math>q_0</math></i>
c.potentialAt( $x$ , $y$ )	<i>electric potential of charge c at point (<math>x, y</math>)</i>
str(c)	<i>'<math>q_0</math> at (<math>x_0, y_0</math>)'</i> (string representation of charge c)

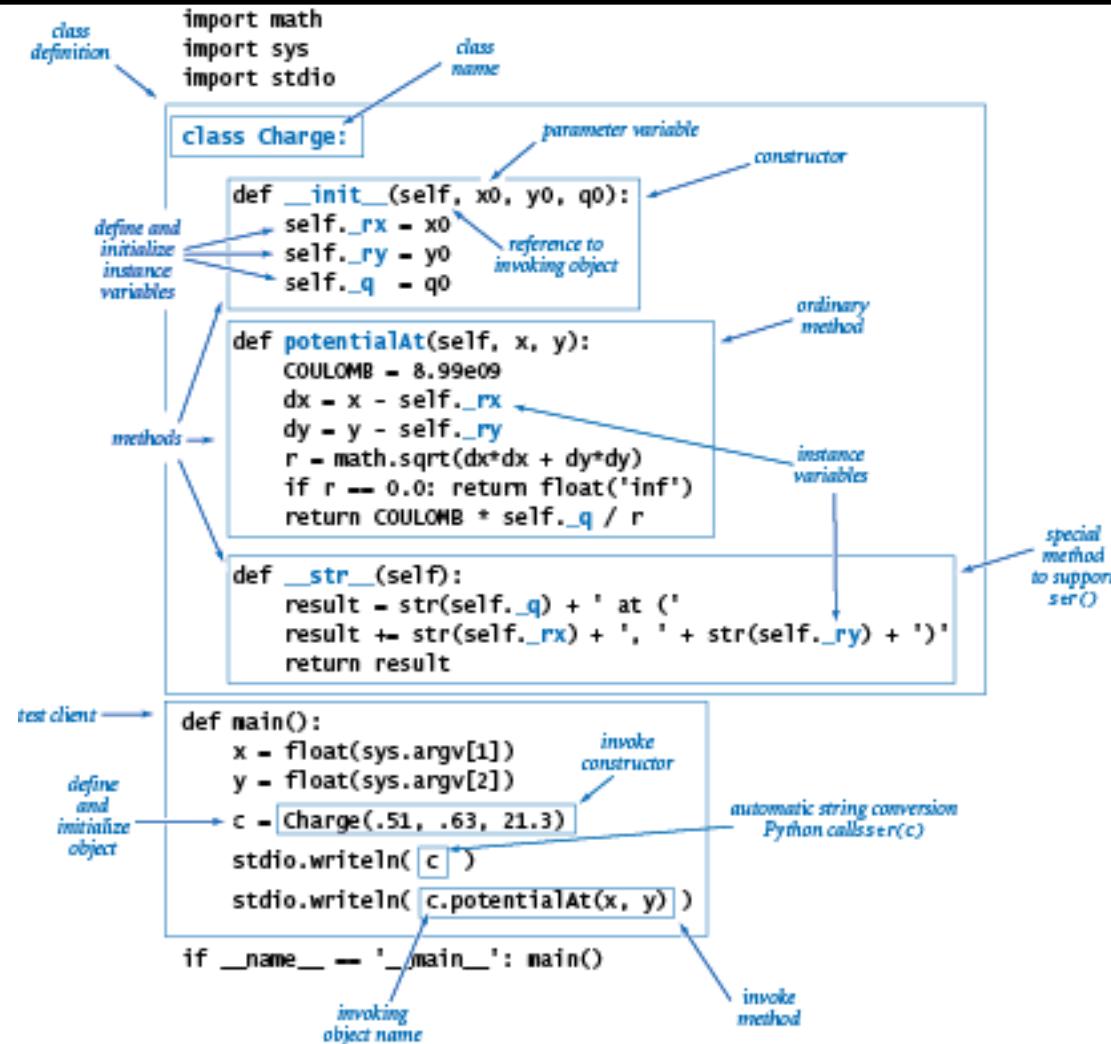
*API for our user-defined Charge data type*

- Clase. Fichero charge.py. Palabra reservada `class`
- Constructor. Método especial `__init__()`, `self`
- Variable de instancia. `_nombrevar`
- Métodos. Variable de instancia `self`
- Funciones intrínsecas. `__str__()`
- Privacidad



# Implementación de Charge

- En charge.py



Anatomy of a class (data-type) definition



# Clases Stopwatch, Histogram, Turtle

- En stopwatch.py

<i>operation</i>	<i>description</i>
<code>Stopwatch()</code>	<i>a new stopwatch (running at the start)</i>
<code>watch.elapsedTime()</code>	<i>elapsed time since watch was created, in seconds</i>

*API for our user-defined Stopwatch data type*

- En histogram.py

<i>operation</i>	<i>description</i>
<code>Histogram(n)</code>	<i>a new histogram from the integer values in 0, 1, ..., n – 1</i>
<code>h.addDataPoint(i)</code>	<i>add an occurrence of integer i to the histogram h</i>
<code>h.draw()</code>	<i>draw h to standard drawing</i>

*API for our user-defined Histogram data type*

- En turtle.py

<i>operation</i>	<i>description</i>
<code>Turtle(x0, y0, a0)</code>	<i>a new turtle at (x0, y0) facing a0 degrees from the x-axis</i>
<code>t.turnLeft(delta)</code>	<i>instruct t to turn left (counterclockwise) by delta degrees</i>
<code>t.goForward(step)</code>	<i>instruct t to move forward distance step, drawing a line</i>

*API for our user-defined Turtle data type*



# Clase Complex

- **Métodos especiales.** En Python la expresión  $a + b$  se reemplaza con la llamada del método `a.__mul__(b)`

<i>client operation</i>	<i>special method</i>	<i>description</i>
<code>Complex(x, y)</code>	<code>__init__(self, re, im)</code>	<i>new Complex object with value <math>x+yi</math></i>
<code>a.re()</code>		<i>real part of a</i>
<code>a.im()</code>		<i>imaginary part of a</i>
<code>a + b</code>	<code>__add__(self, other)</code>	<i>sum of a and b</i>
<code>a * b</code>	<code>__mul__(self, other)</code>	<i>product of a and b</i>
<code>abs(a)</code>	<code>__abs__(self)</code>	<i>magnitude of a</i>
<code>str(a)</code>	<code>__str__(self)</code>	<i>'<math>x + yi</math>' (string representation of a)</i>

*API for a user-defined Complex data type*



# Métodos especiales

<i>client operation</i>	<i>special method</i>	<i>description</i>
<code>x + y</code>	<code>__add__(self, other)</code>	<i>sum of x and y</i>
<code>x - y</code>	<code>__sub__(self, other)</code>	<i>difference of x and y</i>
<code>x * y</code>	<code>__mul__(self, other)</code>	<i>product of x and y</i>
<code>x ** y</code>	<code>__pow__(self, other)</code>	<i>x to the yth power</i>
<code>x / y</code>	<code>__truediv__(self, other)</code>	<i>quotient of x and y</i>
<code>x // y</code>	<code>__floordiv__(self, other)</code>	<i>floored quotient of x and y</i>
<code>x % y</code>	<code>__mod__(self, other)</code>	<i>remainder when dividing x by y</i>
<code>+x</code>	<code>__pos__(self)</code>	<i>x</i>
<code>-x</code>	<code>__neg__(self)</code>	<i>arithmetic negation of x</i>

*Note: Python 2 uses `_div_` instead of `_truediv_`.*

*Special methods for arithmetic operators*



---

# Ficheros

## Tema 6



# Ficheros

- Python dispone de funciones integradas para gestionar la entrada/salida desde ficheros:
  - `open(filename_str, mode)`: retorna un objeto fichero. Los valores válidos de mode son: 'r' (read-only, default), 'w' (write - erase all contents for existing file), 'a' (append), 'r+' (read and write). También se puede usar 'rb', 'wb', 'ab', 'rb+' para operaciones modo binario (raw bytes).
  - `file.close()`: Cierra el objeto file.
  - `file.readline()`: lee una línea (up to a newline and including the newline). Retorna una cadena vacía después end-of-file (EOF).



# Ficheros

- 
- `file.read()`: lee el fichero entero. Retorna una cadena vacía después de end-of-file (EOF).
  - `file.write(str)`: escribe la cadena dada en el fichero.
  - `file.tell()`: retorna la “posición en curso”. La “posición en curso” es el número de bytes desde el inicio del fichero en modo binario, y un número opaco en modo texto.
  - `file.seek(offset)`: asigna la “posición en curso” a offset desde el inicio del fichero.



# Ficheros - Ejemplos

```
>>> f = open('test.txt', 'w')      # Create (open) a file for write  
>>> f.write('apple\n')           # Write given string to file  
>>> f.write('orange\n')  
>>> f.close()                  # Close the file  
  
>>> f = open('test.txt', 'r')    # Create (open) a file for read (default)  
>>> f.readline()               # Read till newline  
'apple\n'  
>>> f.readline()  
'orange\n'  
>>> f.readline()               # Return empty string after end-of-file  
''  
>>> f.close()
```



# Ficheros - Ejemplos

```
>>> f = open('test.txt', 'r')
>>> f.read()                      # Read entire file
'apple\norange\n'
>>> f.close()
>>> f = open('test.txt', 'r') # Test tell() and seek()
>>> f.tell()
0
>>> f.read()
'apple\norange\n'
>>> f.tell()
13
>>> f.read()
 ''
>>> f.seek(0)    # Rewind
0
>>> f.read()
'apple\norange\n'
>>> f.close()
```



# Iterando a través de ficheros

- Se puede procesar un fichero texto línea a línea mediante un for-in-loop

```
withopen('test.txt') as f: # Auto close the file upon exit
    for line in f:
        line = line.rstrip() # Strip trailing spaces and newl
        print(line)
```

```
# Same as above
f = open('test.txt', 'r')
for line in f:
    print(line.rstrip())
f.close()
```



# Iterando a través de ficheros

- Cada línea incluye un `newline`

```
>>> f = open('temp.txt', 'w')
```

```
>>> f.write('apple\n')
```

```
6
```

```
>>> f.write('orange\n')
```

```
7
```

```
>>> f.close()
```

```
>>> f = open('temp.txt', 'r')
```

```
# Line includes a newline, disable print()'s default newln
```

```
>>> for line in f: print(line, end='')
```

```
apple
```

```
orange
```

```
>>> f.close()
```



---

# Estructura datos complejos

## Tema 7



# Creación de lista y diccionario

- 
- Existe una forma concisa para generar una lista (comprehension). Sintaxis:

```
result_list =[expression_with_item for item in in_list]
# with an optional test
result_list =[expression_with_item for item in in_list if test]
# Same as
result_list = []
for item in in_list:
    if test:
        result_list.append(item)
```



# Creación de lista y diccionario

- Ejemplos listas:

```
>>> sq =[item * item for item in range(1,11)]
```

```
>>> sq
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
>>> x =[3, 4, 1, 5]
```

```
>>> sq_x =[item * item for item in x]      # no test, all items
```

```
>>> sq_x
```

```
[9, 16, 1, 25]
```

```
>>> sq_odd =[item * item for item in x if item % 2 != 0]
```

```
>>> sq_odd
```

```
[9, 1, 25]
```

```
# Nested for
```

```
>>> [(x, y) for x in range(1,3) for y in range(1,4) if x != y]
```

```
[(1, 2), (1, 3), (2, 1), (2, 3)]
```



# Creación de lista y diccionario

- Ejemplos diccionarios:

```
# Dictionary {k1:v1, k2:v2,...}  
>>> d={x:x**2 for x in range(1, 5)} # Use braces for dictionary  
>>> d  
{1: 1, 2: 4, 3: 9, 4: 16}
```

```
# Set {v1, v2,...}  
>>> s ={i for i in 'hello' if i not in 'aeiou'}      # Use braces  
>>> s  
{'h', 'l'}
```



# Ciclos – patrones

<i>write first n+1 powers of 2</i>	<pre>power = 1 for i in range(n+1):     stdio.writeln(str(i) + ' ' + str(power))     power *= 2</pre>
<i>write largest power of 2 less than or equal to n</i>	<pre>power = 1 while 2*power &lt;= n:     power *= 2 stdio.writeln(power)</pre>
<i>write a sum <math>(1 + 2 + \dots + n)</math></i>	<pre>total = 0 for i in range(1, n+1):     total += i stdio.writeln(total)</pre>
<i>write a product <math>(n! = 1 \times 2 \times \dots \times n)</math></i>	<pre>product = 1 for i in range(1, n+1):     product *= i stdio.writeln(product)</pre>
<i>write a table of n+1 function values</i>	<pre>for i in range(n+1):     stdio.write(str(i) + ' ')     stdio.writeln(2.0 * math.pi * i / n)</pre>
<i>write the ruler function (see Program 1.2.1)</i>	<pre>ruler = '1' stdio.writeln(ruler) for i in range(2, n+1):     ruler = ruler + ' ' + str(i) + ' ' + ruler stdio.writeln(ruler)</pre>



# Ciclos anidados

## Nested Loop Examples

Nested Loops	Output	Explanation
<pre>for i in range(3) :     for j in range(4) :         print("*", end="")     print()</pre>	***** ***** *****	Prints 3 rows of 4 asterisks each.
<pre>for i in range(4) :     for j in range(3) :         print("*", end="")     print()</pre>	*** *** *** ***	Prints 4 rows of 3 asterisks each.
<pre>for i in range(4) :     for j in range(i + 1) :         print("*", end="")     print()</pre>	* ** *** ****	Prints 4 rows of lengths 1, 2, 3, and 4.



# Ciclos anidados

## Nested Loop Examples

```
for i in range(3) :  
    for j in range(5) :  
        if j % 2 == 1 :  
            print("*", end="")  
        else :  
            print("-", end="")  
    print()
```

-\*-  
-\*-\*  
-\*-\*

Prints alternating dashes and asterisks.

```
for i in range(3) :  
    for j in range(5) :  
        if i % 2 == j % 2 :  
            print("*", end="")  
        else :  
            print(" ", end="")  
    print()
```

\* \* \*  
\* \*  
\* \* \*

Prints a checkerboard pattern.



# Listas

---

- Es una estructura de datos que almacena una secuencia de objetos, normalmente del mismo tipo.
- El acceso a los elementos de la lista se basa en índices encerrados por corchetes. En una lista bidimensional se realiza con un par de índices.
- El índice del primer elemento es 0
- Las formas de procesar arrays en Python son:
  - Tipo de dato implícito Python.
  - Uso del módulo Python numpy.
  - Uso del módulo stdarray.



# Listas - ejemplo

```
x = [0.30, 0.60, 0.10]
y = [0.50, 0.10, 0.40]
total = 0.0
for i in range(len(x)):
    total+= x[i]*y[i]
```

i	x[i]	y[i]	x[i]*y[i]	total
				0.00
0	0.30	0.50	0.15	0.15
1	0.60	0.10	0.06	0.21
2	0.10	0.40	0.04	0.25

*Trace of dot product computation*



# Operaciones y funciones comunes con Listas

## Common List Functions and Operators

Operation	Description
<code>[]</code> <code>[<i>elem</i><sub>1</sub>, <i>elem</i><sub>2</sub>, ..., <i>elem</i><sub><i>n</i></sub>]</code>	Creates a new empty list or a list that contains the initial elements provided.
<code>len(<i>l</i>)</code>	Returns the number of elements in list <i>l</i> .
<code>list(<i>sequence</i>)</code>	Creates a new list containing all elements of the sequence.
<code><i>values</i> * <i>num</i></code>	Creates a new list by replicating the elements in the <i>values</i> list <i>num</i> times.
<code><i>values</i> + <i>moreValues</i></code>	Creates a new list by concatenating elements in both lists.



# Operaciones y funciones comunes con Listas

## | Common List Functions and Operators

Operation	Description
$l[\text{from} : \text{to}]$	Creates a sublist from a subsequence of elements in list $l$ starting at position <code>from</code> and going through but not including the element at position <code>to</code> . Both <code>from</code> and <code>to</code> are optional. (See Special Topic 6.2.)
<code>sum(<math>l</math>)</code>	Computes the sum of the values in list $l$ .
<code>min(<math>l</math>)</code> <code>max(<math>l</math>)</code>	Returns the minimum or maximum value in list $l$ .
$l_1 == l_2$	Tests whether two lists have the same elements, in the same order.



# Métodos comunes con Listas

## Common List Methods

Method	Description
<i>l.pop()</i> <i>l.pop(position)</i>	Removes the last element from the list or from the given position. All elements following the given position are moved up one place.
<i>l.insert(position, element)</i>	Inserts the <i>element</i> at the given <i>position</i> in the list. All elements at and following the given position are moved down.
<i>l.append(element)</i>	Appends the <i>element</i> to the end of the list.
<i>l.index(element)</i>	Returns the position of the given <i>element</i> in the list. The element must be in the list.
<i>l.remove(element)</i>	Removes the given <i>element</i> from the list and moves all elements following it up one position.
<i>l.sort()</i>	Sorts the elements in the list from smallest to largest.



# Matriz con Listas - lectura

```
def lee_matriz(M):
    #Dato de la dimensión de la matriz,
    print('Lectura Matriz')
    m = int(input('Número de filas '))
    n = int(input('Número de columnas '))
    #Creacion matriz nula en invocacion
    #    M = []
    for i in range(m):
        M.append([0]* n)
    #lectura de elementos
    for i in range(m):
        for j in range(n):
            M[i][j] = float(input('In gresa elemento\
({0},{1}): '.format(i,j)))
```



# Matriz con Listas - output

---

```
def imp_matriz(M):
#imprime matriz
    print ('\nMatrix')
    m = len(M)
    n = len(M[0])
    for i in range(m):
        for j in range(n):
            print(M[i][j] , end='\t')
        print('')
```



# Example Problem: Simple Statistics

---

Many programs deal with large collections of similar information.

Words in a document

Students in a course

Data from an experiment

Customers of a business

Graphics objects drawn on the screen

Cards in a deck



# Sample Problem: Simple Statistics

---

Let's review some code we wrote in chapter 8:

```
# average4.py
#     A program to average a set of numbers
#     Illustrates sentinel loop using empty string as sentinel

def main():
    sum = 0.0
    count = 0
    xStr = raw_input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = raw_input("Enter a number (<Enter> to quit) >> ")
    print "\nThe average of the numbers is", sum / count

main()
```



## Sample Problem: Simple Statistics

---

This program allows the user to enter a sequence of numbers, but the program itself doesn't keep track of the numbers that were entered – it only keeps a running total.

Suppose we want to extend the program to compute not only the mean, but also the median and standard deviation.



# Sample Problem: Simple Statistics

---

The *median* is the data value that splits the data into equal-sized parts.

For the data 2, 4, 6, 9, 13, the median is 6, since there are two values greater than 6 and two values that are smaller.

One way to determine the median is to store all the numbers, sort them, and identify the middle value.



## Sample Problem: Simple Statistics

---

The *standard deviation* is a measure of how spread out the data is relative to the mean.

If the data is tightly clustered around the mean, then the standard deviation is small. If the data is more spread out, the standard deviation is larger.

The standard deviation is a yardstick to measure/express how exceptional the data is.



# Sample Problem: Simple Statistics

---

The standard deviation is

$$s = \sqrt{\frac{\sum (\bar{x} - x_i)^2}{n-1}}$$

Here  $\bar{x}$  is the mean,  $x_i$  represents the  $i^{th}$  data value and  $n$  is the number of data values.

The expression  $(\bar{x} - x_i)^2$  is the square of the “deviation” of an individual item from the mean.

$$(\bar{x} - x_i)^2$$



# Sample Problem: Simple Statistics

---

The numerator is the sum of these squared “deviations” across all the data.

Suppose our data was 2, 4, 6, 9, and 13.

The mean is 6.8

The numerator of the standard deviation is

$$(6.8 - 2)^2 + (6.8 - 4)^2 + (6.8 - 6)^2 + (6.8 - 9)^2 + (6.8 - 13)^2 = 149.6$$

$$s = \sqrt{\frac{149.6}{5-1}} = \sqrt{37.4} = 6.11$$



# Sample Problem: Simple Statistics

---

As you can see, calculating the standard deviation not only requires the mean (which can't be calculated until all the data is entered), but also each individual data element! We need some way to remember these values as they are entered.



# Applying Lists

---

We need a way to store and manipulate an entire collection of numbers.

We can't just use a bunch of variables, because we don't know many numbers there will be.

What do we need? Some way of combining an entire collection of values into one object.



# Lists and Arrays

---

Python lists are ordered sequences of items. For instance, a sequence of  $n$  numbers might be called  $S$ :

$$S = s_0, s_1, s_2, s_3, \dots, s_{n-1}$$

Specific values in the sequence can be referenced using subscripts. By using numbers as subscripts, mathematicians can succinctly summarize computations over items in a sequence using subscript variables.

$$\sum_{i=0}^{n-1} s_i$$



# Lists and Arrays

---

Suppose the sequence is stored in a variable `s`. We could write a loop to calculate the sum of the items in the sequence like this:

```
sum = 0
for i in range(n) :
    sum = sum + s[i]
```

Almost all computer languages have a sequence structure like this, sometimes called an *array*.



# Lists and Arrays

---

A list or array is a sequence of items where the entire sequence is referred to by a single name (i.e. `s`) and individual items can be selected by indexing (i.e. `s[i]`). In other programming languages, arrays are generally a fixed size, meaning that when you create the array, you have to specify how many items it can hold. Arrays are generally also *homogeneous*, meaning they can hold only one data type.



# Lists and Arrays

---

Python lists are dynamic. They can grow and shrink on demand.

Python lists are also *heterogeneous*, a single list can hold arbitrary data types.

Python lists are mutable sequences of arbitrary objects.



# List Operations

---

Operator	Meaning
$<\text{seq}> + <\text{seq}>$	Concatenation
$<\text{seq}> * <\text{int-expr}>$	Repetition
$<\text{seq}>[]$	Indexing
<code>len(&lt;seq&gt;)</code>	Length
$<\text{seq}>[:]$	Slicing
<code>for &lt;var&gt; in &lt;seq&gt;:</code>	Iteration
$<\text{expr}> \text{ in } <\text{seq}>$	Membership (Boolean)



# List Operations

---

Except for the membership check, we've used these operations before on strings. The membership operation can be used to see if a certain value appears anywhere in a sequence.

```
>>> lst = [1, 2, 3, 4]
```

```
>>> 3 in lst
```

```
True
```



# List Operations

---

The summing example from earlier can be written like this:

```
sum = 0
for x in s:
    sum = sum + x
```

**Unlike strings, lists are mutable:**

```
>>> lst = [1,2,3,4]
>>> lst[3]
4
>>> lst[3] = "Hello"
>>> lst
[1, 2, 3, 'Hello']
>>> lst[2] = 7
>>> lst
[1, 2, 7, 'Hello']
```



# List Operations

---

A list of identical items can be created using the repetition operator. This command produces a list containing 50 zeroes:

```
zeroes = [0] * 50
```



# List Operations

---

Lists are often built up one piece at a time using append.

```
nums = []
x = input('Enter a number: ')
while x >= 0:
    nums.append(x)
    x = input('Enter a number: ')
```

Here, `nums` is being used as an accumulator, starting out empty, and each time through the loop a new value is tacked on.



# List Operations

---

Method	Meaning
<code>&lt;list&gt;.append(x)</code>	Add element x to end of list.
<code>&lt;list&gt;.sort()</code>	Sort (order) the list. A comparison function may be passed as a parameter.
<code>&lt;list&gt;.reverse()</code>	Reverse the list.
<code>&lt;list&gt;.index(x)</code>	Returns index of first occurrence of x.
<code>&lt;list&gt;.insert(i, x)</code>	Insert x into list at index i.
<code>&lt;list&gt;.count(x)</code>	Returns the number of occurrences of x in list.
<code>&lt;list&gt;.remove(x)</code>	Deletes the first occurrence of x in list.
<code>&lt;list&gt;.pop(i)</code>	Deletes the ith element of the list and returns its value.



# List Operations

```
>>> lst = [3, 1, 4, 1, 5, 9]
>>> lst.append(2)
>>> lst
[3, 1, 4, 1, 5, 9, 2]
>>> lst.sort()
>>> lst
[1, 1, 2, 3, 4, 5, 9]
>>> lst.reverse()
>>> lst
[9, 5, 4, 3, 2, 1, 1]
>>> lst.index(4)
2
>>> lst.insert(4, "Hello")
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1, 1]
>>> lst.count(1)
2
>>> lst.remove(1)
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1]
>>> lst.pop(3)
3
>>> lst
[9, 5, 4, 'Hello', 2, 1]
```



# List Operations

---

Most of these methods don't return a value – they change the contents of the list in some way.

Lists can grow by appending new items, and shrink when items are deleted. Individual items or entire slices can be removed from a list using the `del` operator.



# List Operations

---

```
>>> myList=[34, 26, 0, 10]
>>> del myList[1]
>>> myList
[34, 0, 10]
>>> del myList[1:3]
>>> myList
[34]
```

`del` isn't a list method, but a built-in operation that can be used on list items.



# List Operations

---

## Basic list principles

A list is a sequence of items stored as a single object.

Items in a list can be accessed by indexing, and sublists can be accessed by slicing.

Lists are mutable; individual items or entire slices can be replaced through assignment statements.



# List Operations

---

Lists support a number of convenient and frequently used methods.  
Lists will grow and shrink as needed.



# Statistics with Lists

---

One way we can solve our statistics problem is to store the data in lists.

We could then write a series of functions that take a list of numbers and calculates the mean, standard deviation, and median.

Let's rewrite our earlier program to use lists to find the mean.



# Statistics with Lists

---

Let's write a function called `getNumbers` that gets numbers from the user.

We'll implement the sentinel loop to get the numbers.

An initially empty list is used as an accumulator to collect the numbers.

The list is returned once all values have been entered.



# Statistics with Lists

---

```
def getNumbers():
    nums = []      # start with an empty list

    # sentinel loop to get numbers
    xStr = raw_input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        nums.append(x)    # add this value to the list
        xStr = raw_input("Enter a number (<Enter> to quit) >> ")
    return nums
```

Using this code, we can get a list of numbers from the user with a single line of code:

```
data = getNumbers()
```



# Statistics with Lists

---

Now we need a function that will calculate the mean of the numbers in a list.

Input: a list of numbers

Output: the mean of the input list

```
def mean(nums):  
    sum = 0.0  
    for num in nums:  
        sum = sum + num  
    return sum / len(nums)
```



# Statistics with Lists

---

The next function to tackle is the standard deviation.

In order to determine the standard deviation, we need to know the mean.

Should we recalculate the mean inside of `stdDev`?

Should the mean be passed as a parameter to `stdDev`?



# Statistics with Lists

---

Recalculating the mean inside of `stdDev` is inefficient if the data set is large. Since our program is outputting both the mean and the standard deviation, let's compute the mean and pass it to `stdDev` as a parameter.



# Statistics with Lists

---

```
def stdDev(nums, xbar):  
    sumDevSq = 0.0  
    for num in nums:  
        dev = xbar - num  
        sumDevSq = sumDevSq + dev * dev  
    return sqrt(sumDevSq / (len(nums) - 1))
```

The summation from the formula is accomplished with a loop and accumulator.

sumDevSq stores the running sum of the squares of the deviations.



# Statistics with Lists

---

We don't have a formula to calculate the median. We'll need to come up with an algorithm to pick out the middle value.

First, we need to arrange the numbers in ascending order.  
Second, the middle value in the list is the median.  
If the list has an even length, the median is the average of the middle two values.



# Statistics with Lists

---

Pseudocode -

sort the numbers into ascending order

if the size of the data is odd:

    median = the middle value

else:

    median = the average of the two middle values

return median



# Statistics with Lists

---

```
def median(nums):  
    nums.sort()  
    size = len(nums)  
    midPos = size / 2  
    if size % 2 == 0:  
        median = (nums[midPos] + nums[midPos-1]) / 2.0  
    else:  
        median = nums[midPos]  
    return median
```



# Statistics with Lists

---

With these functions, the main program is pretty simple!

```
def main():
    print 'This program computes mean, median and standard deviation.'

    data = getNumbers()
    xbar = mean(data)
    std = stdDev(data, xbar)
    med = median(data)

    print '\nThe mean is', xbar
    print 'The standard deviation is', std
    print 'The median is', med
```



# Statistics with Lists

---

Statistical analysis routines might come in handy some time, so let's add the capability to use this code as a module by adding:

```
if __name__ == '__main__': main()
```



# Lists of Objects

---

All of the list examples we've looked at so far have involved simple data types like numbers and strings.

We can also use lists to store more complex data types, like our student information from chapter ten.



# Lists of Objects

---

Our grade processing program read through a file of student grade information and then printed out information about the student with the highest GPA.

A common operation on data like this is to sort it, perhaps alphabetically, perhaps by credit-hours, or even by GPA.



# Lists of Objects

---

Let's write a program that sorts students according to GPA using our `Student` class from the last chapter.

Get the name of the input file from the user

Read student information into a list

Sort the list by GPA

Get the name of the output file from the user

Write the student information from the list into a file



# Lists of Objects

---

Let's begin with the file processing. The following code reads through the data file and creates a list of students.

```
def readStudents(filename):
    infile = open(filename, 'r')
    students = []
    for line in infile:
        students.append(makeStudent(line))
    infile.close()
    return students
```

We're using the makeStudent from the gpa program, so we'll need to remember to import it.



# Lists of Objects

---

Let's also write a function to write the list of students back to a file.

Each line should contain three pieces of information, separated by tabs: name, credit hours, and quality points.

```
def writeStudents(students, filename):  
    # students is a list of Student objects  
    outfile = open(filename, 'w')  
    for s in students:  
        outfile.write("%s\t%f\t%f\n" %  
                     (s.getName(), s.getHours(), s.getQPoints()))  
    outfile.close()
```



## Lists of Objects

---

Using the functions `readStudents` and `writeStudents`, we can convert our data file into a list of students and then write them back to a file. All we need to do now is sort the records by GPA. In the statistics program, we used the `sort` method to sort a list of numbers. How does Python sort lists of objects?



# Lists of Objects

---

Python compares items in a list using the built-in function `cmp`.

`cmp` takes two parameters and returns `-1` if the first comes before the second, `0` if they're equal, and `1` if the first comes after the second.



# Lists of objects

---

```
>>> cmp(1, 2)
-1
>>> cmp(2, 1)
1
>>> cmp("a", "b")
-1
>>> cmp(1, 1.0)
0
>>> cmp("a", 5)
1
```

For types that aren't directly comparable, the standard ordering uses rules like “numbers always comes before strings.”



# Lists of Objects

---

To make sorting work with our objects, we need to tell `sort` how the objects should be compared.

We do this by writing our own custom `cmp`-like function and then tell `sort` to use it when sorting.

To sort by GPA, we need a routine that will take two students as parameters and then returns `-1`, `0`, or `1`.



# Lists of Objects

---

We can use the built-in `cmp` function.

```
def cmpGPA(s1, s2):  
    return cmp(s1.gpa(), s2.gpa())
```

We can now sort the data by calling `sort` with the appropriate comparison function (`cmpGPA`) as a parameter.

```
data.sort(cmpGPA)
```



# Lists of Objects

---

```
data.sort(cmpGPA)
```

Notice that we didn't put ()'s after the function call cmpGPA.

This is because we don't want to *call* cmpGPA, but rather, we want to send cmpGPA to the sort method to use as a comparator.



# Lists of Objects

---

```
# gpasort.py
# A program to sort student information into GPA order.

from gpa import Student, makeStudent

def readStudents(filename):
    infile = open(filename, 'r')
    students = []
    for line in infile:
        students.append(makeStudent(line))
    infile.close()
    return students

def writeStudents(students, filename):
    outfile = open(filename, 'w')
    for s in students:
        outfile.write("%s\t%f\t%f\n" %
                      (s.getName(), s.getHours(),
                       s.getQPoints()))
    outfile.close()
```

```
def cmpGPA(s1, s2):
    return cmp(s1.gpa(), s2.gpa())

def main():
    print "This program sorts student grade information by
GPA"
    filename = raw_input("Enter the name of the data file: ")
    data = readStudents(filename)
    data.sort(cmpGPA)
    filename = raw_input("Enter a name for the output file: ")
    writeStudents(data, filename)
    print "The data has been written to", filename

if __name__ == '__main__':
    main()
```



# Designing with Lists and Classes

---

In the `dieView` class from chapter ten, each object keeps track of seven circles representing the position of pips on the face of the die.

Previously, we used specific instance variables to keep track of each, `pip1`, `pip2`, `pip3`,  
...



# Designing with Lists and Classes

---

What happens if we try to store the circle objects using a list?

In the previous program, the pips were created like this:

```
self.pip1 = self.__makePip(cx, cy)
```

`__makePip` is a local method of the `DieView` class that creates a circle centered at the position given by its parameters.



# Designing with Lists and Classes

---

One approach is to start with an empty list of pips and build up the list one pip at a time.

```
pips = []
pips.append(self.__makePip(cx-offset, cy-offset))
pips.append(self.__makePip(cx-offset, cy))
...
self.pips = pips
```



# Designing with Lists and Classes

---

An even more straightforward approach is to create the list directly.

```
self.pips = [self.__makePip(cx-offset, cy-offset),  
            self.__makePip(cx-offset, cy),  
            ...  
            self.__makePip(cx+offset, cy+offset)  
]
```

Python is smart enough to know that this object is continued over a number of lines, and waits for the ']'. Listing objects like this, one per line, makes it much easier to read.



# Designing with Lists and Classes

---

Putting our pips into a list makes many actions simpler to perform.

To blank out the die by setting all the pips to the background color:

```
for pip in self.pips:  
    pip.setFill(self.background)
```

This cut our previous code from seven lines to two!



# Designing with Lists and Classes

---

We can turn the pips back on using the `pips` list. Our original code looked like this:

```
self.pip1.setFill(self.foreground)  
self.pip4.setFill(self.foreground)  
self.pip7.setFill(self.foreground)
```

Into this:

```
self.pips[0].setFill(self.foreground)  
self.pips[3].setFill(self.foreground)  
self.pips[6].setFill(self.foreground)
```



# Designing with Lists and Classes

---

Here's an even easier way to access the same methods:

```
for i in [0,3,6]:  
    self.pips[i].setFill(self.foreground)
```

We can take advantage of this approach by keeping a list of which pips to activate!

Loop through pips and turn them all off

Determine the list of pip indexes to turn on

Loop through the list of indexes - turn on those pips



# Designing with Lists and Classes

---

```
for pip in self.pips:  
    self.pip.setFill(self.background)  
if value == 1:  
    on = [3]  
elif value == 2:  
    on = [0,6]  
elif value == 3:  
    on = [0,3,6]  
elif value == 4:  
    on = [0,2,4,6]  
elif value == 5:  
    on = [0,2,3,4,6]  
else:  
    on = [0,1,2,3,4,5,6]  
for i in on:  
    self.pips[i].setFill(self.foreground)
```



# Designing with Lists and Classes

---

We can do even better!

The correct set of pips is determined by value. We can make this process *table-driven* instead.

We can use a list where each item on the list is itself a list of pip indexes.

For example, the item in position 3 should be the list [0, 3, 6] since these are the pips that must be turned on to show a value of 3.



# Designing with Lists and Classes

---

Here's the table-driven code:

```
onTable = [ [], [3], [2,4], [2,3,4], [0,2,4,6],  
           [0,2,3,4,6], [0,1,2,4,5,6] ]  
  
for pip in self.pips:  
    self.pip.setFill(self.background)  
  
on = onTable[value]  
for i in on:  
    self.pips[i].setFill(self.foreground)
```



# Designing with Lists and Classes

---

```
onTable = [ [], [3], [2,4], [2,3,4], [0,2,4,6], [0,2,3,4,6], [0,1,2,4,5,6] ]  
  
for pip in self.pips:  
    self.pip.setFill(self.background)  
  
on = onTable[value]  
for i in on:  
    self.pips[i].setFill(self.foreground)
```

The table is padded with ‘[]’ in the 0 position, since it shouldn’t ever be used.

The `onTable` will remain unchanged through the life of a `dieView`, so it would make sense to store this table in the constructor and save it in an instance variable.



# Designing with Lists and Classes

---

```
# dieview2.py
#     A widget for displaying the value of a die.
#     This version uses lists to simplify keeping track of
#     pips.

class DieView:
    """ DieView is a widget that displays a graphical
    representation
        of a standard six-sided die."""

    def __init__(self, win, center, size):
        """Create a view of a die, e.g.:
            d1 = GDie(myWin, Point(40,50), 20)
        creates a die centered at (40,50) having sides
        of length 20."""
        # first define some standard values
        self.win = win
        self.background = "white" # color of die face
        self.foreground = "black" # color of the pips
        self.psize = 0.1 * size # radius of each pip
        hsize = size / 2.0 # half of size
        offset = 0.6 * hsize # distance from center to
        outer pips

        # create a square for the face
        cx, cy = center.getX(), center.getY()
        p1 = Point(cx-hsize, cy-hsize)
        p2 = Point(cx+hsize, cy+hsize)
        rect = Rectangle(p1,p2)
        rect.draw(win)
        rect.setFill(self.background)

    # Create 7 circles for standard pip locations
    self.pips = [ self.__makePip(cx-offset, cy-offset),
                  self.__makePip(cx-offset, cy),
                  self.__makePip(cx-offset, cy+offset),
                  self.__makePip(cx, cy),
                  self.__makePip(cx+offset, cy-offset),
                  self.__makePip(cx+offset, cy),
                  self.__makePip(cx+offset, cy+offset) ]

    # Create a table for which pips are on for each value
    self.onTable = [ [], [3], [2,4], [2,3,4],
                    [0,2,4,6], [0,2,3,4,6], [0,1,2,4,5,6] ]

    self.setValue(1)

    def __makePip(self, x, y):
        """Internal helper method to draw a pip at (x,y)"""
        pip = Circle(Point(x,y), self.psize)
        pip.setFill(self.background)
        pip.setOutline(self.background)
        pip.draw(self.win)
        return pip

    def setValue(self, value):
        """ Set this die to display value."""
        # Turn all the pips off
        for pip in self.pips:
            pip.setFill(self.background)

        # Turn the appropriate pips back on
        for i in self.onTable[value]:
            self.pips[i].setFill(self.foreground)
```



# Designing with Lists and Classes

---

Lastly, this example showcases the advantages of encapsulation.

We have improved the implementation of the `dieView` class, but we have not changed the set of methods it supports.

We can substitute this **new** version of the class without having to modify any other code!

Encapsulation allows us to build complex software systems as a set of “pluggable modules.”



---

# Numpy

## Tema 9



# What is Numpy?

---

Numpy, Scipy, and Matplotlib provide MATLAB-like functionality in python.

Numpy Features:

- Typed multidimensional arrays (matrices)
- Fast numerical computations (matrix math)
- High-level math functions



# Why do we need NumPy

---

Let's see for ourselves!





# Why do we need NumPy

---

Python does numerical computations slowly.

1000 x 1000 matrix multiply

Python triple loop takes > 10 min.

Numpy takes ~0.03 seconds



# Logistics: Versioning

---

In this class, your code will be tested with:

Python 2.7.6

Numpy version: 1.8.2

Scipy version: 0.13.3

OpenCV version: 2.4.8

Two easy options:

Class virtual machine (always test on the VM)

Anaconda 2 (some assembly required)

---



# NumPy Overview

---

1. Arrays
  2. Shaping and transposition
  3. Mathematical Operations
  4. Indexing and slicing
  5. Broadcasting
-



---

Structured lists of numbers.

Vectors

Matrices

Images

Tensors

ConvNets



---

Structured lists of numbers.

**Vectors**

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

**Matrices**

Images

Tensors

ConvNets

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

Structured lists of numbers.

Vectors

Matrices

**Images**

Tensors

ConvNets



Structured lists of numbers.

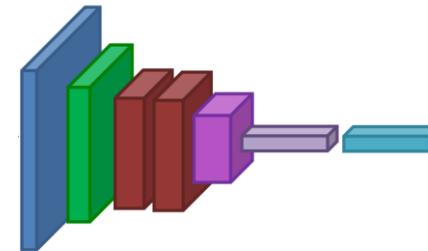
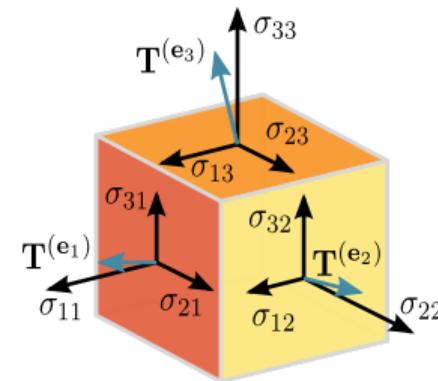
Vectors

Matrices

Images

Tensors

ConvNets



Structured lists of numbers.

Vectors

Matrices

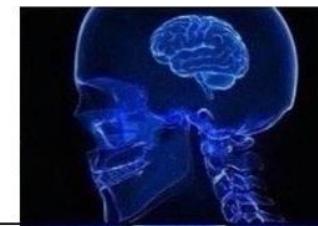
Images

Tensors

ConvNets

**MATRICES**

---



**IMAGES**

---



**TENSORS**

---



**CONVNETS**

---





# Arrays, Basic Properties

---

```
import numpy as np  
a = np.array([[1,2,3], [4,5,6]], dtype=np.float32)  
print a.ndim, a.shape, a.dtype
```

1. Arrays can have any number of dimensions, including zero (a scalar).
  2. Arrays are typed: np.uint8, np.int64, np.float32, np.float64
  3. Arrays are dense. Each element of the array exists and has the same type.
-



# Arrays, creation

---

np.ones, np.zeros  
np.arange  
np.concatenate  
np.astype  
np.zeros\_like, np.ones\_like  
np.random.random



# Arrays, creation

---

**np.ones, np.zeros**

**np.arange**

**np.concatenate**

**np.astype**

**np.zeros\_like, np.ones\_like**

**np.random.random**

```
>>> np.ones((3,5),dtype=np.float32)
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]], dtype=float32)
```

```
>>> np.zeros((6,2),dtype=np.int8)
array([[0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0]], dtype=int8)
```



# Arrays, creation

---

np.ones, np.zeros

**np.arange**

np.concatenate

np.astype

np.zeros\_like, np.ones\_like

np.random.random

```
>>> np.arange(1334,1338)
array([1334, 1335, 1336, 1337])
```



# Arrays, creation

---

np.ones, np.zeros

np.arange

**np.concatenate**

np.astype

np.zeros\_like, np.ones\_like

np.random.random

```
>>> A = np.ones((2,3))
>>> B = np.zeros((4,3))
>>> np.concatenate([A,B])
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>>
```



# Arrays, creation

---

np.ones, np.zeros

np.arange

**np.concatenate**

np.astype

np.zeros\_like, np.ones\_like

np.random.random

```
>>> A = np.ones((4,1))
>>> B = np.zeros((4,2))
>>> np.concatenate([A,B], axis=1)
array([[ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.]])
```



# Arrays, creation

---

np.ones, np.zeros

np.arange

np.concatenate

**np.astype**

np.zeros\_like, np.ones\_like

np.random.random

```
>>> A
array([[ 4670.5,   4670.5,   4670.5],
       [ 4670.5,   4670.5,   4670.5],
       [ 4670.5,   4670.5,   4670.5],
       [ 4670.5,   4670.5,   4670.5],
       [ 4670.5,   4670.5,   4670.5]], dtype=float32)
>>> print(A.astype(np.uint16))
[[4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]
 [4670 4670 4670]]
```



# Arrays, creation

---

np.ones, np.zeros  
np.arange  
np.concatenate  
np.astype  
**np.zeros\_like, np.ones\_like**  
np.random.random

```
>>> a = np.ones((2,2,3))
>>> b = np.zeros_like(a)
>>> print(b.shape)
```



# Arrays, creation

---

np.ones, np.zeros

np.arange

np.concatenate

np.astype

np.zeros\_like, np.ones\_like

**np.random.random**

```
>>> np.random.random((10,3))
array([[ 0.61481644,  0.55453657,  0.04320502],
       [ 0.08973085,  0.25959573,  0.27566721],
       [ 0.84375899,  0.2949532 ,  0.29712833],
       [ 0.44564992,  0.37728361,  0.29471536],
       [ 0.71256698,  0.53193976,  0.63061914],
       [ 0.03738061,  0.96497761,  0.01481647],
       [ 0.09924332,  0.73128868,  0.22521644],
       [ 0.94249399,  0.72355378,  0.94034095],
       [ 0.35742243,  0.91085299,  0.15669063],
       [ 0.54259617,  0.85891392,  0.77224443]])
```



# Arrays, danger zone

---

Must be dense, no holes.

Must be one type

Cannot combine arrays of different shape

```
>>> np.ones([7,8]) + np.ones([9,3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together
          with shapes (7,8) (9,3)
```



```
a = np.array([1,2,3,4,5,6])
a = a.reshape(3,2)
a = a.reshape(2,-1)
a = a.ravel()
1. Total number of elements cannot change.
2. Use -1 to infer axis shape
3. Row-major by default (MATLAB is column-major)
```



# Return values

---

Numpy functions return either **views** or **copies**.

Views share data with the original array, like references in Java/C++. Altering entries of a view, changes the same entries in the original.

The [numpy documentation](#) says which functions return views or copies

`Np.copy`, `np.view` make explicit copies and views.



# Transposition

---

```
a = np.arange(10).reshape(5, 2)
a = a.T
a = a.transpose((1, 0))
```

np.transpose permutes axes.

a.T transposes the first two axes.



# Saving and loading arrays

---

```
np.savez('data.npz', a=a)
data = np.load('data.npz')
a = data['a']
```

1. NPZ files can hold multiple arrays
  2. np.savez\_compressed similar.
-



# Image arrays

---

Images are 3D arrays: width, height, and channels

Common image formats:

height x width x RGB (band-interleaved)

height x width (band-sequential)



Gotchas:

Channels may also be BGR (OpenCV does this)

May be [width x height], not [height x width]





# Saving and Loading Images

---

SciPy: `skimage.io.imread, skimage.io.imsave`  
height x width x RGB

PIL / Pillow: `PIL.Image.open, Image.save`  
width x height x RGB

OpenCV: `cv2.imread, cv2.imwrite`  
height x width x BGR

---



---

We just saw how to create arrays, reshape them, and permute axes

Questions so far?



---

We just saw how to create arrays, reshape them, and permute axes

Questions so far?

Now: let's do some math



# Mathematical operators

---

Arithmetic operations are element-wise

Logical operator return a bool array

In place operations modify the array



# Mathematical operators

---

**Arithmetic operations are element-wise**

Logical operator return a bool array

In place operations modify the array

```
>>> a  
array([1, 2, 3])  
>>> b  
array([ 4,  4, 10])  
>>> a * b  
array([ 4,  8, 30])
```



# Mathematical operators

---

Arithmetic operations are element-wise

**Logical operator return a bool array**

In place operations modify the array

```
>>> a
array([[ 0.93445601,  0.42984044,  0.12228461],
       [ 0.06239738,  0.76019703,  0.11123116],
       [ 0.14617578,  0.90159137,  0.89746818]])
>>> a > 0.5
array([[ True, False, False],
       [False,  True, False],
       [False,  True,  True]], dtype=bool)
```



# Mathematical operators

---

Arithmetic operations are element-wise

Logical operator return a bool array

**In place operations modify the array**

```
>>> a  
array([[ 4, 15],  
       [20, 75]])  
>>> b  
array([[ 2,  5],  
       [ 5, 15]])  
>>> a /= b  
>>> a  
array([[2, 3],  
       [4, 5]])
```



# Math, upcasting

---

Just as in Python and Java, the result of a math operator is cast to the more general or precise datatype.

`uint64 + uint16 => uint64`

`float32 / int32 => float32`

Warning: upcasting does not prevent overflow/underflow. You must manually cast first.

Use case: images often stored as `uint8`. You should convert to `float32` or `float64` before doing math.

---



# Math, universal functions

---

Also called ufuncs

Element-wise

Examples:

`np.exp`

`np.sqrt`

`np.sin`

`np.cos`

`np.isnan`





# Math, universal functions

---

Also called ufuncs

Element-wise

Examples:

`np.exp`

`np.sqrt`

`np.sin`

`np.cos`

`np.isnan`





# Math, universal functions

---

Also called ufuncs

Element-wise

Examples:

np.exp

np.sqrt

np.sin

np.cos

np.isnan

```
>>> a  
array([[ 1,  4],  
       [ 9, 16],  
       [25, 36]])  
>>> np.sqrt(a)  
array([[ 1.,  2.],  
       [ 3.,  4.],  
       [ 5.,  6.]])
```



# Indexing

---

```
x[0,0]      # top-left element  
x[0,-1]     # first row, last column  
x[0,:] # first row (many entries)  
x[:,0] # first column (many entries)
```

Notes:

Zero-indexing

Multi-dimensional indices are comma-separated (i.e., a tuple)



# Indexing, slices and arrays

---

```
I[1:-1,1:-1]                      # select all but one-  
pixel border  
I = I[:, :, ::-1]                 # swap channel order  
I[I<10] = 0                       # set dark pixels to black  
I[[1,3], :]                      # select 2nd and 4th row
```

1. Slices are **views**. Writing to a slice overwrites the original array.
  2. Can also index by a list or boolean array.
-



# Python Slicing

---

Syntax: start:stop:step

```
a = list(range(10))
a[:3] # indices 0, 1, 2
a[-3:] # indices 7, 8, 9
a[3:8:2] # indices 3, 5, 7
a[4:1:-1] # indices 4, 3, 2 (this one is tricky)
```



---

```
a.sum() # sum all entries  
a.sum(axis=0) # sum over rows  
a.sum(axis=1) # sum over columns  
a.sum(axis=1, keepdims=True)
```

1. Use the axis parameter to control which axis NumPy operates on
2. Typically, the axis specified will disappear, keepdims keeps all dimensions



# Broadcasting

---

```
a = a + 1 # add one to every element
```

When operating on multiple arrays, broadcasting rules are used.

Each dimension must match, from right-to-left

1. Dimensions of size 1 will broadcast (as if the value was repeated).
2. Otherwise, the dimension must have the same shape.
3. Extra dimensions of size 1 are added to the left as needed.



# Broadcasting example

---

Suppose we want to add a color value to an image

a.shape is 100, 200, 3

b.shape is 3

a + b will pad b with two extra dimensions so it has an effective shape of  $1 \times 1 \times 3$ .

So, the addition will broadcast over the first and second dimensions.

---



# Broadcasting failures

---

If `a.shape` is `100, 200, 3` but `b.shape` is `4` then `a + b` will fail. The trailing dimensions must have the same shape (or be 1)





# Tips to avoid bugs

---

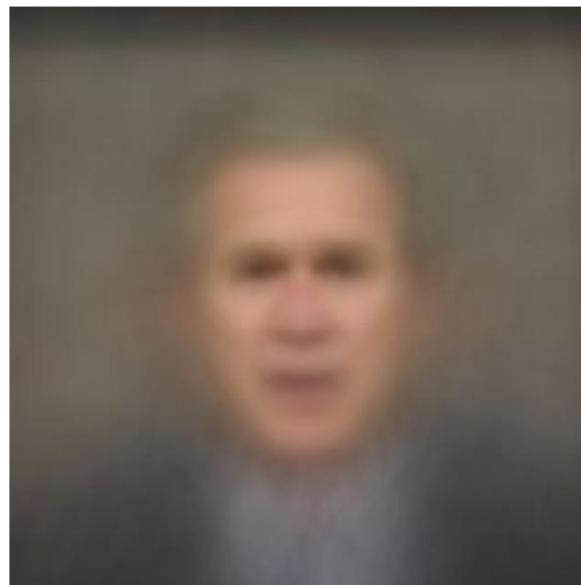
1. Know what your datatypes are.
  2. Check whether you have a view or a copy.
  3. Use matplotlib for sanity checks.
  4. Use pdb to check each step of your computation.
  5. Know np.dot vs np.mult.
-



# Average images

---

Who is this?





# Practice exercise (not graded)

---

Compute the average image of faces.

1. Download Labeled Faces in the Wild dataset (google: LFW face dataset). Pick a face with at least 100 images.
2. Call `numpy.zeros` to create a  $250 \times 250 \times 3$  float64 tensor to hold the result
3. Read each image with `skimage.io.imread`, convert to float and accumulate
4. Write the averaged result with `skimage.io.imsave`



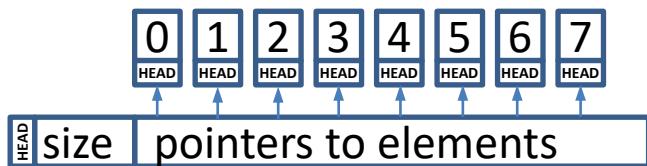
---

# Pandas

## Tema 10

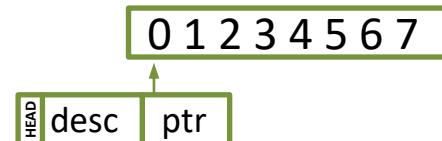
# python vs numpy – size

```
>>> import sys  
>>> gso = sys.getsizeof  
  
>>> lst = list(range(1000))  
[ 0, 1, 2, 3, 4, 5, 6, 7, ... ]
```



```
>>> gso(0)          # 24 bytes  
>>> gso([])         # 64 bytes  
>>> gso(lst)        # 9112 bytes  
>>> gso(lst) + sum(gso(x) for x in lst)  
                      # 37108 bytes  
  
>>> hex(id(lst))    # '0x7f1e9c07ed48'
```

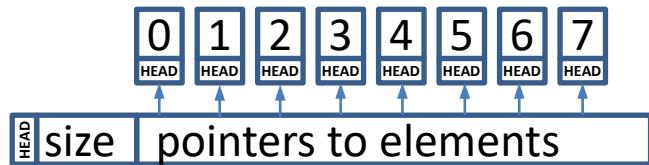
```
>>> import numpy as np  
  
>>> arr = np.arange(1000, dtype=np.int32)  
array([ 0, 1, 2, 3, 4, 5, 6, 7, ... ], dtype=int32)
```



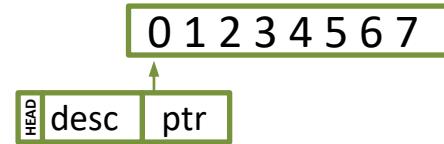
```
>>> gso(np.array([], dtype=np.int32)) # 96 bytes  
>>> arr.itemsize                  # 4  
>>> arr nbytes                   # 4000  
>>> gso(arr)                     # 4096 bytes  
  
>>> arr.data          # <memory at 0x7f1e9c14ab88>  
>>> arr.shape, arr.strides      # (1000,), (4,)
```

# python vs numpy – speed

```
>>> lst = list(range(1000000))  
[ 0, 1, 2, 3, 4, 5, 6, 7, ... ]
```



```
>>> arr = np.arange(1000000, dtype='i4')  
array([ 0, 1, 2, 3, 4, 5, 6, 7, ... ], dtype=int32)
```



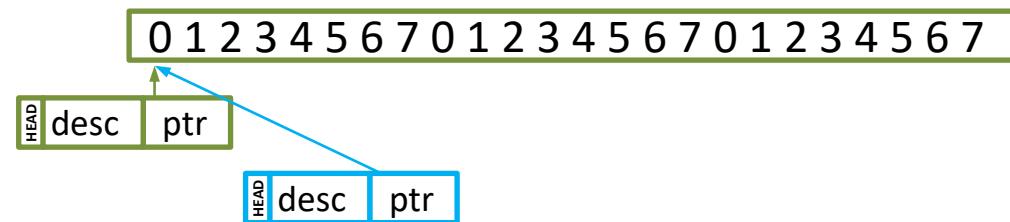
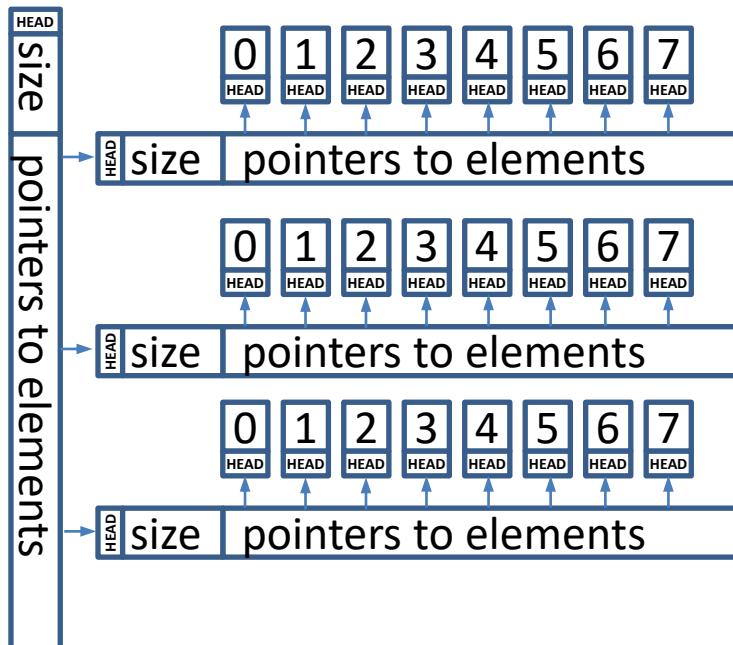
```
>>> %timeit sum(lst)      # 10 ms
```

```
>>> %timeit arr.sum()    # 1.2 ms
```

```
>>> %timeit np.sum(arr)  # 1.2 ms
```

```
>>> %timeit sum(arr)    # 150 ms
```

# python vs numpy – 2D data



```
>>> arr = np.arange(24, dtype='i4')  
  
>>> arr2 = arr.reshape((3,8))  
  
array([[ 0,  1,  2,  3,  4,  5,  6,  7],  
       [ 8,  9, 10, 11, 12, 13, 14, 15],  
       [16, 17, 18, 19, 20, 21, 22, 23]], dtype=int32)
```

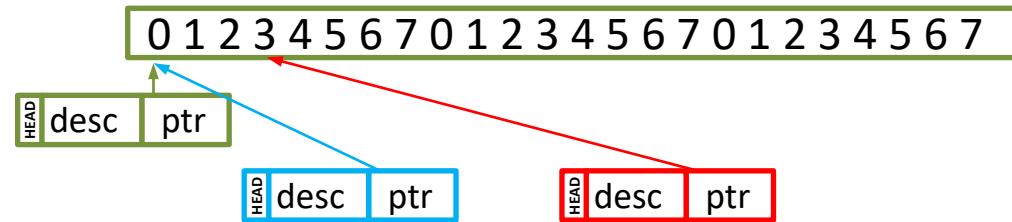
# numpy – slicing and views

```
>>> arr = np.arange(24, dtype='i4')
>>> arr2 = arr.reshape((3,8))
>>> arr3 = arr[3::3]
```

>>> arr3.base is arr

True

```
>>> np.info(arr), np.info(arr2), np.info(arr3)
class: ndarray
shape: (24,) (3,8) (7,)
strides: (4,) (32,4) (12,)
itemsize: 4
aligned: True
contiguous: True True False
fortran: True False False
data pointer: 0x1df3690 0x1df3690 0x1df369c
byteorder: little
byteswap: False
type: int32
```



# numpy – indexing

---

```
>>> arr = np.arange(24, dtype='i4')  
>>> arr2 = arr.reshape((3,8))  
array([[ 0,  1,  2,  3,  4,  5,  6,  7],  
      [ 8,  9, 10, 11, 12, 13, 14, 15],  
      [16, 17, 18, 19, 20, 21, 22, 23]], dtype=int32)
```

## Scalar index

```
>>> arr2[1]  
array([ 8,  9, 10, 11, 12, 13, 14, 15], dtype=int32)
```

## Slice

```
>>> arr3 = arr2[:,0:4]  
array([[ 0,  1,  2,  3],  
      [ 8,  9, 10, 11],  
      [16, 17, 18, 19]], dtype=int32)
```

## Integer row indexes (location)

```
>>> arr4 = arr3.ravel()  
>>> arr4[ [1,5,7] ]  
>>> arr4.take([1,5,7])  
array([0,1,2,3,8,9,10,11,16,17,18,19], dtype=int32)  
array([ 1,  9, 11], dtype=int32)  
array([ 1,  9, 11], dtype=int32)
```

## Boolean indexing

```
>>> arr4 % 3 == 0  
>>> arr4[ arr4 % 3 == 0 ]  
array([ True, False, False, True, False, ...], dtype=bool)  
array([ 0,  3,  9, 18], dtype=int32)
```

## Assigning to a slice

```
>>> arr4[ arr4 % 3 == 0 ] = -1  
array([-1,1,2,-1,8,-1,10,11,16,17,-1,19], dtype=int32)
```



# Pandas = best of Python + numpy + R

---

Python

- Easy syntax
- Good for prototyping (“...but slow”)
- Helpful community

Numpy

- Fast, memory-efficient calcs
- Well-tested algorithms

R

- DataFrame column labels
- Indexes to align rows



# Top-level classes

---

`DataFrame`

`Series`

`TimeSeries`

`Index`

`MultiIndex`

`Int64Index`

`Float64Index`

`RangeIndex`

`Grouper`

`Categorical`

`CategoricalIndex`

`Timestamp`

`DatetimeIndex`

`Timedelta`

`TimedeltaIndex`

`Period`

`PeriodIndex`

`DateOffset`

`TimeGrouper`

`Panel / WidePanel / Panel4D`

`ExcelFile / ExcelWriter / HDFStore`

`SparseArray / SparseDataFrame / SparseSeries`

`Expr / Term`



# Top-level classes

# core.internals

DataFrame  
Series  
TimeSeries

Index  
MultiIndex  
Int64Index  
Float64Index  
RangeIndex

Grouper  
Categorical  
CategoricalIndex

Timestamp  
DatetimeIndex  
Timedelta  
TimedeltaIndex  
Period  
PeriodIndex  
DateOffset  
TimeGrouper

Panel / WidePanel / Panel4D  
ExcelFile / ExcelWriter / HDFStore  
SparseArray / SparseDataFrame / SparseSeries  
Expr / Term

SingleBlockManager  
BlockManager  
BlockPlacement  
  
JoinUnit  
NonConsolidatableMixIn  
  
Block  
ObjectBlock / BoolBlock  
NumericBlock / IntBlock  
FloatBlock / ComplexBlock  
DatetimeBlock /  
TimeDeltaBlock  
DatetimeTZBlock /  
DatetimeTZDtype  
CategoricalBlock  
SparseBlock



# Top-level classes

# core.internals

DataFrame  
Series  
TimeSeries

Index  
MultiIndex  
Int64Index  
Float64Index  
RangeIndex

Grouper  
Categorical  
CategoricalIndex

Timestamp  
DatetimeIndex  
Timedelta  
TimedeltaIndex  
Period  
PeriodIndex  
DateOffset  
TimeGrouper

Panel / WidePanel / Panel4D  
ExcelFile / ExcelWriter / HDFStore  
SparseDataFrame / SparseSeries  
Expr / Term

# pd.DataFrame

(444 methods!)

T, abs, add, add\_prefix, add\_suffix, align, all, any, append, apply, applymap, as\_blocks, as\_matrix, asfreq, assign, astype, at, at\_time, axes, between\_time, bfill, blocks, bool, boxplot, clip, clip\_lower, clip\_upper, columns, combine, combineAdd, combineMult, combine\_first, compound, consolidate, convert\_objects, copy, corr, corrwith, count, cov, cummax, cummin, cumprod, cumsum, describe, diff, div, divide, dot, drop, drop\_duplicates, dropna, dtypes, duplicated, empty, eq, equals, eval, ewm, expanding, ffill, fillna, filter, first, first\_valid\_index, floordiv, from\_csv, from\_dict, from\_items, from\_records, ftypes, ge, get, get\_dtype\_counts, get\_ftype\_counts, get\_value, get\_values, groupby, gt, head, hist, iat, icol, idxmax, idxmin, iget\_value, iloc, index, info, insert, interpolate, irow, is\_copy, isin, isnull, items, iteritems, iterkv, iterrows, iter tuples, ix, join, keys, kurt, kurtosis, last, last\_valid\_index, le, loc, lookup, lt, mad, mask, max, mean, median, merge, min, mod, mode, mul, multiply, ndim, ne, nlargest, notnull, nsmallest, pct\_change, pipe, pivot, pivot\_table, plot, pop, pow, prod, product, quantile, query, radd, rank, rdiv, reindex, reindex\_axis, reindex\_like, rename, rename\_axis, reorder\_levels, replace, resample, reset\_index, rfloordiv, rmod, rmul, rolling, round, rpow, rsub, rtruediv, sample, select, select\_dtypes, sem, set\_axis, set\_index, set\_value, shape, shift, size, skew, slice\_shift, sort, sort\_index, sort\_values, sortlevel, squeeze, stack, std, style, sub, subtract, sum, swapaxes, swaplevel, tail, take, to\_clipboard, to\_csv, to\_dense, to\_dict, to\_excel, to\_gbq, to\_hdf, to\_html, to\_json, to\_latex, to\_msgpack, to\_panel, to\_period, to\_pickle, to\_records, to\_sparse, to\_sql, to\_stata, to\_string, to\_timestamp, to\_wide, to\_xarray, transpose, truediv, truncate, tshift, tz\_convert, tz\_localize, unstack, update, values, var, where, xs ...  
\_AXIS\_ALIASES, \_AXIS\_SLICEMAP, \_AXIS\_ORDER, \_AXIS\_NUMBERS, \_AXIS\_NAMES, \_AXIS\_LEN, \_ACCESSORS, \_CONSTRUCTORS, \_CONTRACTORS, \_CREATE\_INDEXERS, \_DIR\_ADDITIONS, \_DIR\_DELETIONS, \_ENSURE\_VALID\_INDEX, \_EXPAND\_AXES, \_FLEX\_COMPARE\_FRAME, \_FROM\_ARRAYS, \_GET\_AXIS, \_GET\_AXIS\_NAME, \_GET\_AXIS\_NUMBER, \_GET\_AXIS\_RESOLVERS, \_GET\_BLOCK\_MANAGER\_AXIS, \_GET\_BOOL\_DATA, \_GET\_CACHER, \_GET\_INDEX\_RESOLVERS, \_GET\_ITEM\_CACHE, \_GET\_NUMERIC\_DATA, \_GET\_VALUES, \_GETITEM\_ARRAY, \_GETITEM\_COLUMN, \_GETITEM\_FRAME, \_GETITEM\_MULTILEVEL, \_GETITEM\_SLICE, \_IAT, \_IGET\_ITEM\_CACHE, \_ILOC, \_INDEXED\_SAME, \_INFO\_AXIS\_NAME, \_INFO\_AXIS\_NUMBER, \_INFO\_REPR, \_INIT\_DICT, \_INIT\_MGR, \_INIT\_NDARRAY, \_INTERNAL\_NAMES, \_INTERNAL\_NAMES\_SET, \_IS\_CACHED, \_IS\_DATELIKE\_MIXED\_TYPE, \_IS\_MIXED\_TYPE, \_IS\_NUMERIC\_MIXED\_TYPE, \_IS\_VIEW, \_IX, \_IXS, \_JOIN\_COMPAT, \_LOC, \_MAYBE\_CACHE\_CHANGED, \_MAYBE\_UPDATE\_CACHER, \_METADATA, \_NEEDS\_REINDEX\_MULTI, \_NSORTED, \_PROTECT\_CONSOLIDATE, \_REDUCE, \_REINDEX\_AXES, \_REINDEX\_COLUMNS, \_REINDEX\_INDEX, \_REINDEX\_MULTI, \_REINDEX\_WITH\_INDEXERS, \_REPR\_FITS\_HORIZONTAL, \_REPR\_FITS\_VERTICAL, \_REPR\_HTML, \_REPR\_LATEX, \_RESET\_CACHE, \_RESET\_CACHER, \_SANITIZE\_COLUMN, \_SERIES, \_SET\_AS\_CACHED, \_SET\_AXIS, \_SET\_AXIS\_NAME, \_SET\_IS\_COPY, \_SET\_ITEM, \_SETITEM\_ARRAY, \_SETITEM\_FRAME, \_SETITEM\_SLICE, \_SETUP\_AXES, \_SLICE, \_STAT\_AXIS, \_STAT\_AXIS\_NAME, \_STAT\_AXIS\_NUMBER, \_TYP, \_UNPICKLE\_FRAME\_COMPAT, \_UNPICKLE\_MATRIX\_COMPAT, \_UPDATE\_INPLACE, \_VALIDATE\_DTYPE, \_VALUES, \_XS

SingleBlockManager

BlockManager

BlockPlacement

JoinUnit

NonConsolidatableMixIn

Block

ObjectBlock / BoolBlock

NumericBlock / IntBlock

FloatBlock / ComplexBlock

DatetimeBlock /  
TimeDeltaBlock

DatetimeTZBlock /  
DatetimeTZDtype

CategoricalBlock

SparseBlock

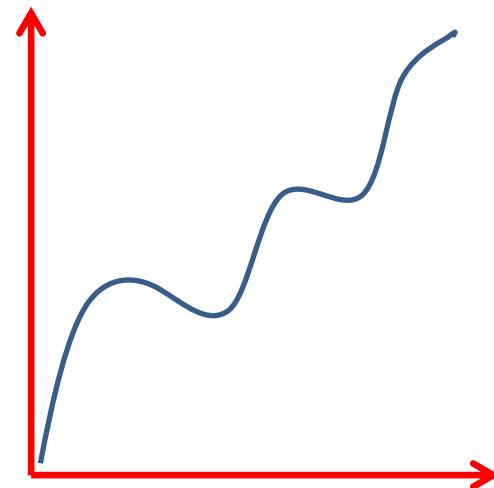
345

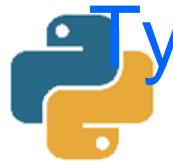


# Pandas = a bumpy learning curve

---

- Broad top-level namespace
- Syntactic sugar
- Large codebase
- Rapid evolution
- API backwards compatible
- Stack Overflow answers
- ‘Big data’





# Typical pandas analytical steps

---

1. Load raw data into DataFrame
2. Reformat columns and add row indexes
3. Select subsets of rows
4. Aggregate and subtotal with GroupBy
5. Post-process for display
6. Compare with other data



# Starting point: Aussie Rules football data

All games in chronological order

```
1. 8-May-1897   R1 Fitzroy     6.13.49 Carlton     2.4.16 Brunswick St
2. 8-May-1897   R1 Collingwood 5.11.41 St Kilda    2.4.16 Victoria Park
3. 8-May-1897   R1 Geelong     3.6.24 Essendon    7.5.47 Corio Oval
...
14838. 1-May-2016  R6 Brisbane Lions 14.10.94 Sydney      15.7.97 Gabba
14839. 1-May-2016  R6 Carlton       10.12.72 Essendon    8.9.57 M.C.G.
14840. 1-May-2016  R6 West Coast   18.16.124 Collingwood 9.8.62 Subiaco
```

**Goals \* 6 + Behinds == Points**

<http://afltables.com/afl/stats/biglists/bg3.txt>

# Aussie Rules = footy = football (!= soccer)

---



# End result: premiership ladder



## Toyota AFL Premiership Season Ladder

Pos		Club	P	W	L	D	F	A	%	Pts
1	North Melbourne	6	6	0	0	672	538	124.9	24	
2	Geelong Cats	6	5	1	0	664	380	174.7	20	
3	Sydney Swans	6	5	1	0	645	448	144	20	
4	Western Bulldogs	6	4	2	0	536	344	155.8	16	
5	GWS Giants	6	4	2	0	670	476	140.8	16	
6	West Coast Eagles	6	4	2	0	609	467	130.4	16	
7	Adelaide Crows	6	4	2	0	688	564	122	16	
8	Hawthorn	6	4	2	0	560	610	91.8	16	
9	Melbourne	6	3	3	0	605	592	102.2	12	
10	Gold Coast Suns	6	3	3	0	565	595	95	12	
11	Port Adelaide	6	3	3	0	549	612	89.7	12	
12	St Kilda	6	2	4	0	553	625	88.5	8	
13	Collingwood	6	2	4	0	501	637	78.7	8	
14	Carlton	6	2	4	0	388	528	73.5	8	
15	Richmond	6	1	5	0	482	646	74.6	4	
16	Brisbane Lions	6	1	5	0	509	719	70.8	4	
17	Essendon	6	1	5	0	363	586	62	4	
18	Fremantle	6	0	6	0	430	622	69.1	0	

<http://www.afl.com.au/ladder>



# -1. Load raw data into DataFrame -

All games in chronological order

```
1. 8-May-1897    R1 Fitzroy      6.13.49 Carlton     2.4.16 Brunswick St
2. 8-May-1897    R1 Collingwood  5.11.41 St Kilda    2.4.16 Victoria Park
3. 8-May-1897    R1 Geelong      3.6.24 Essendon    7.5.47 Corio Oval
...
14838. 1-May-2016   R6 Brisbane Lions 14.10.94 Sydney      15.7.97 Gabba
14839. 1-May-2016   R6 Carlton       10.12.72 Essendon    8.9.57 M.C.G.
14840. 1-May-2016   R6 West Coast    18.16.124 Collingwood 9.8.62 Subiaco
```

```
cols = 'GameNum Date Round HomeTeam HomeScore AwayTeam AwayScore Venue'
df = pd.read_csv(filename, skiprows=2, sep='[. ] +',
                 names=cols.split(), parse_dates=['Date'],
                 quoting=csv.QUOTE_NONE, engine='python')
```

351

```
# Or using the sample tutorial code:
>>> import pfi
>>> df = pfi.load_data('bg3.txt')
```

# DataFrame structure

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

[14840 rows x 8 columns]

```
>>> df.index      # RangeIndex(start=0, stop=14840, step=1)
>>> df.columns    # Index(['GameNum','Date','Round','HomeTeam',...], dtype='object')
>>> df.dtypes.tolist()  # [ dtype('int64'), dtype('<M8[ns]'), dtype('O'), dtype('O'), ...]
>>> df.values      # numpy.array or numpy.recarray
```

```
array([[1, Timestamp('1897-05-08 00:00:00'), 'R1', ..., '2.4.16', 'Brunswick St'],
       [2, Timestamp('1897-05-08 00:00:00'), 'R1', ..., '2.4.16', 'Victoria Park'],
       [3, Timestamp('1897-05-08 00:00:00'), 'R1', ..., '7.5.47', 'Corio Oval'],
       ...,
       [14838, Timestamp('2016-05-01 00:00:00'), 'R6', ..., '15.7.97', 'Gabba'],
       [14839, Timestamp('2016-05-01 00:00:00'), 'R6', ..., '8.9.57', 'M.C.G.'],
       [14840, Timestamp('2016-05-01 00:00:00'), 'R6', ..., '9.8.62', 'Subiaco']])
], dtype=object)
```



# Selecting DataFrame columns

---

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gaoba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

[14840 rows x 8 columns]

```
>>> %timeit df.values          # 35ms - numpy.recarray
```

```
>>> %timeit df[['Round', 'HomeScore']]    # 500µs – pandas.DataFrame
```

```
>>> %timeit df['Round'], df['HomeScore']    # 2µs per col – tuple of pandas.Series
```

353

```
>>> %timeit df['HomeScore']           # 2µs - pandas.Series
```

```
>>> %timeit df[['HomeScore']]        # 480µs – pandas.DataFrame
```



# Behind the scenes: BlockManager

```
>>> df
```

```
   GameNum      Date Round HomeTeam HomeScore AwayTeam AwayScore   Venue
0    1 1897-05-08   R1   Fitzroy  6.13.49 Carlton  2.4.16 Brunswick St
1    2 1897-05-08   R1 Collingwood 5.11.41 St Kilda  2.4.16 Victoria Park
2    3 1897-05-08   R1   Geelong  3.6.24 Essendon  7.5.47 Corio Oval
...
14837 4838 2016-05-01   R6 Brisbane Lions 14.10.94 Sydney 15.7.97 Gabba
14838 4839 2016-05-01   R6 Carlton 10.12.72 Essendon 8.5.57 M.C.G.
14839 4840 2016-05-01   R6 West Coast 18.16.124 Collingwood 9.8.62 Subiaco
[14840 rows x 8 columns]
```

```
>>> df.blocks # Dict with columns with same dtype -> homogeneous numpy array
```

```
{'object': <DF with obj cols>, 'int64': <DF with int64 cols>, 'datetime64[ns]': <DF with TS cols>}
```

```
>>> bm = df._data # BlockManager
```

```
>>> bm.blocks
```

```
(IntBlock: slice(0, 1, 1), 1 x 14840, dtype: int64,
 DatetimeBlock: slice(1, 2, 1), 1 x 14840, dtype: datetime64[ns],
 ObjectBlock: slice(2, 8, 1), 6 x 14840, dtype: object)
```

```
>>> bm._blkno, bm._blklocs
```

```
(array([0, 1, 2, 2, 2, 2, 2]), array([0, 0, 0, 1, 2, 3, 4, 5]))
```

```
>>> df.columns
```

```
Index(['GameNum', 'Date', 'Round', 'HomeTeam', 'HomeScore', 'AwayTeam', 'AwayScore', 'Venue'], dtype='object')
```

```
>>> %timeit -n10 df._data.blocks[2].values[2]
```

```
array(['6.13.49', '5.11.41', '3.6.24', ..., '14.10.94', '10.12.72', '18.16.124'], dtype=object)
```

```
10 loops, best of 3: 338 ns per loop
```

```
>>> df['HomeScore'].values.base is df._data.blocks[2].values[2].base
```

```
True
```



# Plus caching: df[col] → df.\_\_getitem\_\_

---

```
>>> df.__getitem__??  
  
def __getitem__(self, key):  
    # shortcut if we are an actual column  
    is_mi_columns = isinstance(self.columns, MultiIndex)  
    try:  
        if key in self.columns and not is_mi_columns:  
            return self._getitem_column(key)  
    except:  
        pass  
  
    # see if we can slice the rows  
    indexer = convert_to_index_sliceable(self, key)  
    if indexer is not None:  
        return self._getitem_slice(indexer)  
  
    if isinstance(key, (Series, np.ndarray, Index, list)):  
        # either boolean or fancy integer index  
        return self._getitem_array(key)  
    elif isinstance(key, DataFrame):  
        return self._getitem_frame(key)  
    elif is_mi_columns:  
        return self._getitem_multilevel(key)  
    else:  
        return self._getitem_column(key)  
  
>>> df._getitem_column??  
  
def _getitem_column(self, key):  
    """ return the actual column """  
    # get column  
    if self.columns.is_unique:  
        return self._get_item_cache(key)  
    # duplicate columns & possible reduce dimensionality  
    result = self._constructor(self._data.get(key))  
    if result.columns.is_unique:  
        result = result[key]  
    return result
```

```
>>> df._get_item_cache??  
  
def _get_item_cache(self, item):  
    """Return the cached item, item represents a label indexer."""  
    cache = self._item_cache  
    res = cache.get(item)  
    if res is None:  
        values = self._data.get(item)  
        res = self._box_item_values(item, values)  
        cache[item] = res  
        res._set_as_cached(item, self)  
    # for a chain  
    res.is_copy = self.is_copy  
    return res
```

```
>>> df._clear_item_cache()
```

```
>>> %timeit -n10 df['GameNum']  
The slowest run took 6.93 times longer than the fastest.  
This could mean that an intermediate result is being cached.  
10 loops, best of 3: 1.95µs per loop
```

```
>>> df._item_cache  
{'GameNum': <series_obj>}
```

```
>>> %timeit -n10 df['GameNum']  
10 loops, best of 3: 1.88µs per loop
```

# Selecting DataFrame columns (2)

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	2	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	3	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

[14840 rows x 8 columns]

```
>>> df[['GameNum', 'Round']] # 500us
```

```
>>> idx = df.columns.get_indexer(['GameNum','Round']) # 120us
array([0, 2])
```

```
>>> df.take(idx, axis=1) # 360us
```

Terminology: **indexer** = array of integers  
saying which items to take

GameNum Round

0	1	R1
1	2	R1
2	3	R1
...	...	...
14837	14838	R6
14838	14839	R6
14839	14840	R6

356

[14840 rows x 2 columns]



# Selecting DataFrame rows

---

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1897-05-08	R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St	
1	1897-05-08	R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park	
2	1897-05-08	R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval	
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

[14840 rows x 8 columns]

```
>>> df2 = df[14837:14839] # See note...
```

```
GameNum Date Round HomeTeam HomeScore AwayTeam AwayScore Venue
14837 14838 2016-05-01 R6 Brisbane Lions 14.10.94 Sydney 15.7.97 Gabba
14838 14839 2016-05-01 R6 Carlton 10.12.72 Essendon 8.9.57 M.C.G.
```

```
>>> %timeit -n10 df[14837:14839]
```

10 loops, best of 3: 127 µs per loop

```
>>> df2.columns is df.columns
```

True

```
>>> df2.index
```

RangeIndex(start=14837, stop=14839, step=1)

Note: compare df['GameNum'][14837:14839]  
and df['GameNum'].iloc[14837:14839]



# Select = index lookup + slice numpy array plus index

```
>>> df
```

	GameNum	Date	Round	HomeTeam	HomeScore	AwayTeam	AwayScore	Venue
0	1897-05-08		R1	Fitzroy	6.13.49	Carlton	2.4.16	Brunswick St
1	1897-05-08		R1	Collingwood	5.11.41	St Kilda	2.4.16	Victoria Park
2	1897-05-08		R1	Geelong	3.6.24	Essendon	7.5.47	Corio Oval
...	...	...	...	...	...	...	...	...
14837	14838	2016-05-01	R6	Brisbane Lions	14.10.94	Sydney	15.7.97	Gabba
14838	14839	2016-05-01	R6	Carlton	10.12.72	Essendon	8.9.57	M.C.G.
14839	14840	2016-05-01	R6	West Coast	18.16.124	Collingwood	9.8.62	Subiaco

[14840 rows x 8 columns]

```
>>> %timeit df[14837:14839]          # 99 µs per loop
```

```
>>> %timeit df['HomeScore'][14837:14839]    # 68 µs per loop - index lookup
```

```
>>> %timeit df['HomeScore'].values[14837:14839] # 3 µs per loop - direct slice of numpy array
```

```
>>> s = df['HomeScore']
```

```
>>> %timeit s[14837:14839]          # 64 µs per loop
```

```
>>> %timeit s.values[14837:14839]    # 936 ns per loop
```



## -2. Reformat columns. Add row indexes

```
>>> df
   GameNum Date Round HomeTeam HomeScore AwayTeam AwayScore      Venue
0     1 1897-05-08 R1    Fitzroy  6.13.49    Carlton  2.4.16 Brunswick St
1     2 1897-05-08 R1 Collingwood 5.11.41   St Kilda  2.4.16 Victoria Park
...
...   ...   ...   ...
14839 14840 2016-05-01 R6 West Coast 18.16.124 Collingwood 9.8.62 Subiaco
[14840 rows x 8 columns]
```

```
# Convert into sections for both teams
dfi = df.set_index(['Date','Venue','Round'])
home_teams = dfi['HomeTeam'].rename('Team')
away_teams = dfi['AwayTeam'].rename('Team')

# Split scores into Goals/Behinds/pts For and Against
regex = '(?P<G>\d+).(?P<B>\d+).(?P<F>\d+)'
home_scores = dfi['HomeScore']
    .str.extract(regex, expand=True).astype(int)
away_scores = dfi['AwayScore']
    .str.extract(regex, expand=True).astype(int)

home_scores['A'] = away_scores['F']
away_scores['A'] = home_scores['F']

# Combine into new DataFrame
home = pd.concat([home_teams, home_scores], axis=1)
away = pd.concat([away_teams, away_scores], axis=1)

scores = home.append(away)
    .set_index('Team', append=True)
    .sort_index()
```

```
>>> import pfi
>>> df = pfi.load_data('bg3.txt')
>>> scores =
    pfi.prepare_game_scores(df)
```

```
>>> scores
          G  B  F  A
Date   Venue Round Team
1897-05-08 Brunswick St  R1    Fitzroy 6 13 49 16
                    R1    Carlton 2 4 16 49
Corio Oval    R1    Geelong 3 6 24 47
                    R1    Essendon 7 5 47 24
Lake Oval     R1 South Melbourne 3 9 27 44
                    R1    Melbourne 6 8 44 27
Victoria Park R1 Collingwood 5 11 41 16
                    R1    St Kilda 2 4 16 41
1897-05-15 East Melbourne R2    Essendon 4 6 30 50
...
...
2016-04-30 M.C.G.     R6    Richmond 8 11 59 94
                    R6 Port Adelaide 13 16 94 59
Sydney Showground R6    GW Sydney 24 14 158 83
                    R6    Hawthorn 12 11 83 158
2016-05-01 Gabba     R6 Brisbane Lions 14 10 94 97
                    R6    Sydney 15 7 97 94
M.C.G.         R6    Carlton 10 12 72 57
                    R6    Essendon 8 9 57 72
Subiaco        R6 West Coast 18 16 124 62
                    R6 Collingwood 9 8 62 124
[29680 rows x 5 columns]
```

# Indexes – looking up values

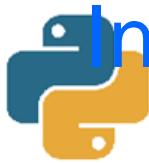


```
>>> i = pd.Index(['a','b','b','c'])

>>> i.is_unique, i.is_monotonic
(False, True)
>>> i.get_loc('a')
0
>>> i.get_loc('b')
slice(1, 3, None)

>>> i = pd.Index(['a','b','c','b'])

>>> i.is_unique, i.is_monotonic
(False, False)
>>> i.get_loc('a')
0
>>> i.get_loc('b')
array([False, True, False, True], dtype=bool)
```



# Indexes – aligning

---

```
>>> a = pd.Series([1,2,3,4],  
                 index=['a','b','c','d'])  
>>> b = pd.Series([5,6,7,8],  
                 index=['a','b','d','e'])
```

```
>>> a, b, a+b
```

```
a    1  
b    2  
c    3  
d    4  
dtype: int64
```

```
a    5  
b    6  
d    7  
e    8  
dtype: int64
```

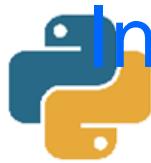
```
a    6.0  
b    8.0  
c    NaN  
d    11.0  
e    NaN  
dtype: float64
```

```
>>> a.index  
Index(['a', 'b', 'c', 'd'], dtype='object')  
>>> b.index  
Index(['a', 'b', 'd', 'e'], dtype='object')  
>>> i = a.index.union(b.index)  
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

```
>>> a.reindex(i)  
a    1.0  
b    2.0  
c    3.0  
d    4.0  
e    NaN  
dtype: float64
```

```
>>> a.reindex(i) + b.reindex(i)  
a    6.0  
b    8.0  
c    NaN  
d    11.0  
e    NaN  
dtype: float64
```

```
>>> b.reindex(i)  
a    5.0  
b    6.0  
c    NaN  
d    7.0  
e    8.0  
dtype: float64
```



# Indexes - Join and merge

---

```
>>> a = pd.DataFrame( [[1,2],[3,4],[5,6],[7,8]], index=['a','b','c','d'], columns=['A1','A2'])
```

```
>>> b = pd.DataFrame( [[11,12],[13,14],[15,16],[17,18]], index=['a','b','d','d'], columns=['B1','B2'])
```

```
>>> a.join(b, how='inner')
```

	A1	A2	B1	B2
a	1	2	11	12
b	3	4	13	14
d	7	8	15	16
d	7	8	17	18

```
>>> a.join(b, how='left')
```

	A1	A2	B1	B2
a	1	2	11.0	12.0
b	3	4	13.0	14.0
c	5	6	NaN	NaN
d	7	8	15.0	16.0
d	7	8	17.0	18.0

```
>>> pandas.tools.merge.merge(a, b, left_index=True, right_index=True, how='left')
```



# Join and merge

```
>>> a = pd.DataFrame( [[1,2],[3,4],[5,6],[7,8]],  
...     index=['a','b','c','d'], columns=['A1','A2'])  
>>> b = pd.DataFrame([[11,12],[13,14],[15,16],[17,18]],  
...     index=['a','b','d','d'], columns=['B1','B2'])  
  
>>> # a.join(b, how='left')  
>>> op = pandas.tools.merge._MergeOperation(  
...     a, b, left_index=True,  
...     right_index=True, how='left')  
>>> op.get_result()  
A1 A2 B1 B2  
a 1 2 11.0 12.0  
b 3 4 13.0 14.0  
c 5 6 NaN NaN  
d 7 8 15.0 16.0  
d 7 8 17.0 18.0  
  
>>> (join_index, left_indexer, right_indexer)  
= op._get_join_info()  
>>> join_index  
Index(['a', 'b', 'c', 'd', 'd'], dtype='object')  
>>> left_indexer  
array([0, 1, 2, 3, 3])  
>>> right_indexer  
array([ 0, 1, -1, 2, 3])
```

```
# DataFrame.join() => DataFrame._join_compat() => pandas.tools.merge.merge()  
def merge(left, right, how='inner', on=None, left_on=None, right_on=None,  
         left_index=False, right_index=False, sort=False,  
         suffixes=('_x', '_y'), copy=True, indicator=False):  
    op = _MergeOperation(left, right, how=how, on=on, left_on=left_on,  
                         right_on=right_on, left_index=left_index,  
                         right_index=right_index, sort=sort, suffixes=suffixes,  
                         copy=copy, indicator=indicator)  
    return op.get_result()  
  
class _MergeOperation(object):  
    """  
        Perform a database (SQL) merge operation between two DataFrame objects  
        using either columns as keys or their row indexes  
    """  
    _merge_type = 'merge'  
    def __init__(self, left, right, how='inner', on=None,  
                 left_on=None, right_on=None, axis=1,  
                 left_index=False, right_index=False, sort=True,  
                 suffixes=('_x', '_y'), copy=True, indicator=False):  
        self.left = self.orig_left = left  
        self.right = self.orig_right = right  
        <snip>  
  
    def get_result(self):  
        join_index, left_indexer, right_indexer = self._get_join_info()  
        ldata, rdata = self.left._data, self.right._data  
        lsuf, rsuf = self.suffixes  
        llables, rlables = items_overlap_with_suffix(ldata.items, lsuf,  
                                                    rdata.items, rsuf)  
        lindexers = {1: left_indexer} if left_indexer is not None else {}  
        rindexers = {1: right_indexer} if right_indexer is not None else {}  
        result_data = concatenate_block_managers(  
            [(ldata, lindexers), (rdata, rindexers)],  
            axes=[llables.append(rlables), join_index],  
            concat_axis=0, copy=self.copy)  
        typ = self.left._constructor  
        result = typ(result_data).__finalize__(self, method=self._merge_type)  
        self._maybe_add_join_keys(result, left_indexer, right_indexer)  
        return result
```



```
>>> scores
```

Date	Venue	Round	Team	G	B	F	A
1897-05-08	Brunswick St	R1	Fitzroy	6	13	49	16
	R1	Carlton	2	4	16	49	
	Corio Oval	R1	Geelong	3	6	24	47
	R1	Essendon	7	5	47	24	
	Lake Oval	R1	South Melbourne	3	9	27	44
	R1	Melbourne	6	8	44	27	
	Victoria Park	R1	Collingwood	5	11	41	16
	R1	St Kilda	2	4	16	41	
1897-05-15	East Melbourne	R2	Essendon	4	6	30	50
...							
2016-04-30	M.C.G.	R6	Richmond	8	11	59	94
	R6	Port Adelaide	13	16	94	59	
	Sydney Showground	R6	GW Sydney	24	14	158	83
	R6	Hawthorn	12	11	83	158	
2016-05-01	Gabba	R6	Brisbane Lions	14	10	94	97
	R6	Sydney	15	7	97	94	
	M.C.G.	R6	Carlton	10	12	72	57
	R6	Essendon	8	9	57	72	
	Subiaco	R6	West Coast	18	16	124	62
	R6	Collingwood	9	8	62	124	

[29680 rows x 5 columns]

```
>>> %timeit df.values
35.3 ms per loop
>>> df._data.blocks
(IntBlock: slice(0, 1, 1), 1 x 14840, dtype: int64,
 DatetimeBlock: slice(1, 2, 1), 1 x 14840, dtype: datetime64[ns],
 ObjectBlock: slice(2, 8, 1), 6 x 14840, dtype: object)
```

```
>>> scores.columns
```

```
Index(['G', 'B', 'F', 'A'], dtype='object')
```

```
>>> mi = scores.index # pd.MultiIndex
```

```
>>> mi.names
```

```
FrozenList(['Date', 'Venue', 'Round', 'Team'])
```

```
>>> mi.levels
```

```
[ DatetimeIndex(['1897-05-08', ..., '2016-05-01'],
```

```
dtype='datetime64[ns]', name='Date', length=4496),
```

```
Index(['Adelaide Oval', 'Albury', 'Arden St',
```

```
'Bellerive Oval', ..., 'York Park'],
```

```
dtype='object', name='Venue'),
```

```
Index(['EF', 'GF', 'PF', 'QF', 'R1', 'R10', 'R11',
```

```
..., 'R6', 'R7', 'R8', 'R9', 'SF'],
```

```
dtype='object', name='Round'),
```

```
Index(['Adelaide', 'Brisbane Bears', 'Brisbane Lions',
```

```
'Carlton', 'Collingwood', 'Essendon', ...],
```

```
dtype='object', name='Team')
```

```
>>> mi.labels
```

```
FrozenList([ [ 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,...], [ 7, 7, 11, 11, 20, 20,
```

```
...],
```

```
[ 4, 4, 4, 4, 4, 4, 4, 15, 15, 15, 15, ...],
```

```
[ 6, 3, 10, 5, 18, 14, 4,...] ])
```

```
>>> scores.values
```

```
array([[ 6,  2,  3, ...,  8, 18,  9],  
       [ 13,  4,  6, ...,  9, 16,  8],  
       [ 49, 16, 24, ..., 57, 124, 62],  
       [ 16, 49, 47, ..., 72, 62, 124]])
```

```
>>> %timeit scores.values
```

```
5.85 µs per loop
```

```
>>> scores._data.blocks
```

```
(IntBlock: slice(0, 4, 1), 4 x 29680, dtype: int64,)
```



## 3. Select subsets of rows

```
>>> scores.loc(axis=0)['2016':'2016', :, 'R1':'R9']
```

Date	Venue	Round	Team	G	B	F	A
2016-03-24	M.C.G.	R1	Richmond	14	8	92	83
			Carlton	12	11	83	92
2016-03-26	Carrara	R1	Gold Coast	17	19	121	60
			Essendon	9	6	60	121
Docklands		R1	North Melbourne	16	11	107	97
			Adelaide	14	13	97	107
M.C.G.		R1	Melbourne	12	8	80	78
			GW Sydney	10	18	78	80
S.C.G.		R1	Sydney	18	25	133	53
			Collingwood	7	11	53	133
...							
2016-04-30	Sydney Showground	R6	GW Sydney	24	14	158	83
			Hawthorn	12	11	83	158
2016-05-01	Gabba	R6	Brisbane Lions	14	10	94	97
			Sydney	15	7	97	94
M.C.G.		R6	Carlton	10	12	72	57
			Essendon	8	9	57	72
Subiaco		R6	West Coast	18	16	124	62
			Collingwood	9	8	62	124

[108 rows x 4 columns]

Note: pandas 0.18.0 has a bug if dates are in level 0  
and either first or last dates are not actually in the index:

```
>>> scores.loc(axis=0)['2015-04-02':'2015-09-06', :, 'R1':'R9'] # 394 rows  
>>> scores.loc(axis=0)['2015-04-02':'2015-09-07', :, 'R1':'R9'] # 24577 rows
```

Seems to work ok if dates are last level:

```
>>> scores2 = scores.reorder_levels([1,2,3,0]).sort_index()  
>>> scores2.loc(axis=0)[:, 'R1':'R9', :, '2015':'2015'] # 394 rows
```

Approach #1 – 820 ms

```
>>> pd.concat( [ sdf for (dt, rnd), sdf in  
scores.groupby(level=['Date', 'Round'])  
if dt.year == 2016 and rnd.startswith('R')  
], axis=0)
```

Approach #2 – 1.2 s

```
>>> keep = lambda key: key[0].year==2016  
and key[2].startswith('R')  
>>> scores[ [ keep(key)  
for key, data in scores.iterrows() ] ]
```

Approach #3 – 8 ms

```
>>> scores[ [ keep(key) for key in scores.index ] ]  
100 loops, best of 3: 8.29 ms per loop
```

Approach #4 – 12 ms

```
>>> glv = scores.index.get_level_values  
>>> scores[ glv('Date').year==2016  
& glv('Round').str.startswith('R') ]
```

Approach #5 – 6 ms

```
>>> scores.loc(axis=0)['2016':'2016', :, 'R1':'R9']
```



# DataFrame.loc[] supports in-place updates!

---

```
>>> scores2 = scores.copy()  
>>> scores2.loc(axis=0)['2016', :, 'R1':'R9'] = 0  
>>> scores2
```

Date	Venue	Round	Team	G	B	F	A
1897-05-08	Brunswick St	R1	Fitzroy	6	13	49	16
		Carlton		2	4	16	49
	Corio Oval	R1	Geelong	3	6	24	47
			Essendon	7	5	47	24
	Lake Oval	R1	South Melbourne	3	9	27	44
			Melbourne	6	8	44	27
	Victoria Park	R1	Collingwood	5	11	41	16
		St Kilda		2	4	16	41
1897-05-15	East Melbourne	R2	Essendon	4	6	30	50
			Collingwood	8	2	50	30
...				..	..	..	..
2016-04-30	M.C.G.	R6	Richmond	0	0	0	0
			Port Adelaide	0	0	0	0
	Sydney Showground	R6	GW Sydney	0	0	0	0
			Hawthorn	0	0	0	0
2016-05-01	Gabba	R6	Brisbane Lions	0	0	0	0
			Sydney	0	0	0	0
	M.C.G.	R6	Carlton	0	0	0	0
			Essendon	0	0	0	0
	Subiaco	R6	West Coast	0	0	0	0
			Collingwood	0	0	0	0

[29680 rows x 4 columns]



# Behind the scenes...

```
>>> scores2 = scores.copy()  
>>> scores2.loc(axis=0)['2016', :, 'R1':'R9'] # 2.0 ms  
>>> scores2.loc(axis=0)['2016', :, 'R1':'R9'] = 0 # 3.3 ms  
>>> scores2
```

Date	Venue	Round	G	B	F	A	
1897-05-08	Brunswick St	R1	Fitzroy	6	13	49	16
	Carlton		2	4	16	49	
Corio Oval	R1	Geelong	3	6	24	47	
	Essendon		7	5	47	24	
Lake Oval	R1	South Melbourne	3	9	27	44	
	Melbourne		6	8	44	27	
Victoria Park	R1	Collingwood	5	11	41	16	
	St Kilda		2	4	16	41	
1897-05-15	East Melbourne	R2	Essendon	4	6	30	50
	Collingwood		8	2	50	30	
...							
2016-04-30	M.C.G.	R6	Richmond	0	0	0	0
	Port Adelaide		0	0	0	0	
Sydney Showground	R6	GW Sydney	0	0	0	0	
	Hawthorn		0	0	0	0	
2016-05-01	Gabba	R6	Brisbane Lions	0	0	0	0
	Sydney		0	0	0	0	
M.C.G.	R6	Carlton	0	0	0	0	
	Essendon		0	0	0	0	
Subiaco	R6	West Coast	0	0	0	0	
	Collingwood		0	0	0	0	

[29680 rows x 4 columns]



```
>>> tup = ( slice('2016','2016'),  
           slice(None),  
           slice('R1','R9') )  
>>> indexer = scores2.index.get_locs(tup) # 1.34 ms
```

```
array([29572, 29573, 29574, 29575, 29576, 29577, 29578, 29579, 29580,  
      29581, 29582, 29583, 29584, 29585, 29586, 29587, 29588, 29589,  
      29590, 29591, 29592, 29593, 29594, 29595, 29596, 29597, 29598,  
      29599, 29600, 29601, 29602, 29603, 29604, 29605, 29606, 29607,  
      29608, 29609, 29610, 29611, 29612, 29613, 29614, 29615, 29616,  
      29617, 29618, 29619, 29620, 29621, 29622, 29623, 29624, 29625,  
      29626, 29627, 29628, 29629, 29630, 29631, 29632, 29633, 29634,  
      29635, 29636, 29637, 29638, 29639, 29640, 29641, 29642, 29643,  
      29644, 29645, 29646, 29647, 29648, 29649, 29650, 29651, 29652,  
      29653, 29654, 29655, 29656, 29657, 29658, 29659, 29660, 29661,  
      29662, 29663, 29664, 29665, 29666, 29667, 29668, 29669, 29670,  
      29671, 29672, 29673, 29674, 29675, 29676, 29677, 29678, 29679])
```

```
>>> scores2.ix[indexer] # 330 µs
```

```
>>> scores2.ix[indexer] = 0 # 190 µs
```

# 4. Add calculated columns

Need to count # of games played, won, drawn, lost

```
>>> y = scores.loc(axis=0)['2016', :, 'R1':'R9']
>>> y['P'] = 1
```

#### *SettingWithCopyWarning:*

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>

```
>>> y.is_copy
<weakref at 0x7f3accf35868;
to 'DataFrame' at 0x7f3ace5e3a90>
```

```
>>> hex(id(scores))
'0x7f3ace5e3a90'
```

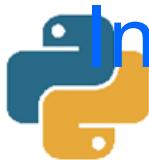
```
>>> y = y.copy()
```

```
>>> y.is_copy
None
```

```
>>> y['P'] = 1    # How fast?
```

Date	Venue	Round	Team	G	B	F	A	P
2016-03-24	M.C.G.	R1	Richmond	14	8	92	83	1
		Carlton		12	11	83	92	1
2016-03-26	Carrara	R1	Gold Coast	17	19	121	60	1
		Essendon		9	6	60	121	1
	Docklands	R1	North Melbourne	16	11	107	97	1
		Adelaide		14	13	97	107	1
	M.C.G.	R1	Melbourne	12	8	80	78	1
				...	..	..	..	..
2016-04-30	Sydney Showground	R6	Hawthorn	12	11	83	158	1
2016-05-01	Gabba	R6	Brisbane Lions	14	10	94	97	1
		Sydney		15	7	97	94	1
	M.C.G.	R6	Carlton	10	12	72	57	1
		Essendon		8	9	57	72	1
	Subiaco	R6	West Coast	18	16	124	62	1
		Collingwood		9	8	62	124	1

[108 rows x 5 columns]



# Inserting a column is fast...

---

```
>>> %timeit -n1 y['P'] = 1
1 loop, best of 3: 142 µs per loop

>>> y._data.blocks
(IntBlock: slice(0, 4, 1), 4 x 108, dtype: int64,
 IntBlock: slice(4, 5, 1), 1 x 108, dtype: int64)

>>> y._data.is_consolidated()
False

>>> _ = y.max()    # Most funcs trigger consolidation

>>> y._data.is_consolidated()
True

>>> y._data.blocks
(IntBlock: slice(0, 5, 1), 5 x 108, dtype: int64,)
```

# Creating the other columns

```
>>> y = y.reset_index(['Date','Venue',
   'Round'], drop=True)

>>> y['P'] = 1
>>> y['W'] = (y['F'] > y['A'])
>>> y.loc[y['F'] == y['A'], 'D'] = 1
>>> y.eval('L = 1*(A>F)', inplace=True)
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	True	NaN	0
Carlton	12	11	83	92	1	False	NaN	1
Gold Coast	17	19	121	60	1	True	NaN	0
Essendon	9	6	60	121	1	False	NaN	1
North Melbourne	16	11	107	97	1	True	NaN	0
Adelaide	14	13	97	107	1	False	NaN	1
Melbourne	12	8	80	78	1	True	NaN	0
GW Sydney	10	18	78	80	1	False	NaN	1
Sydney	18	25	138	53	1	True	NaN	0
Collingwood	7	11	63	133	1	False	NaN	1
Port Adelaide	20	13	133	100	1	True	NaN	0
...	...	...	...	...	...	...	...	...
Gold Coast	7	6	48	168	1	False	NaN	1
Richmond	8	11	59	94	1	False	NaN	1
Port Adelaide	13	16	94	59	1	True	NaN	0
GW Sydney	24	14	158	83	1	True	NaN	0
Hawthorn	12	11	13	158	1	False	NaN	1
Brisbane Lions	14	10	94	97	1	False	NaN	1
Sydney	15	7	97	94	1	True	NaN	0
Carlton	10	12	72	57	1	True	NaN	0
Essendon	8	9	57	72	1	False	NaN	1
West Coast	18	16	124	62	1	True	NaN	0
Collingwood	9	8	62	124	1	False	NaN	1

[108 rows x 8 columns]

# Creating the other columns (2)

```
>>> y = y.reset_index(['Date','Venue','Round'], drop=True)
```

```
>>> y['P'] = 1  
>>> y['W'] = (y['F'] > y['A'])  
>>> y.loc[y['F'] == y['A'], 'D'] = 1  
>>> y.eval('L = 1*(A>F)', inplace=True)
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	True	NaN	0
Carlton	12	11	83	92	1	False	NaN	1
Gold Coast	17	19	121	60	1	True	NaN	0
Essendon	9	6	60	121	1	False	NaN	1
North Melbourne	16	11	107	97	1	True	NaN	0
Adelaide	14	13	97	107	1	False	NaN	1
Melbourne	12	8	80	78	1	True	NaN	0
GW Sydney	10	18	78	80	1	False	NaN	1
Sydney	18	25	138	53	1	True	NaN	0
Collingwood	7	11	63	133	1	False	NaN	1
Port Adelaide	20	13	133	100	1	True	NaN	0
...	...	...	...	...	...	...	...	...
Gold Coast	7	6	48	168	1	False	NaN	1
Richmond	8	11	59	94	1	False	NaN	1
Port Adelaide	13	16	94	59	1	True	NaN	0
GW Sydney	24	14	158	83	1	True	NaN	0
Hawthorn	12	11	63	158	1	False	NaN	1
Brisbane Lions	14	10	94	97	1	False	NaN	1
Sydney	15	7	97	94	1	True	NaN	0
Carlton	10	12	72	57	1	True	NaN	0
Essendon	8	9	57	72	1	False	NaN	1
West Coast	18	16	124	62	1	True	NaN	0
Collingwood	9	8	62	124	1	False	NaN	1

[108 rows x 8 columns]

# Creating the other columns (3)

```
>>> y = y.reset_index(['Date','Venue','Round'], drop=True)
```

```
>>> y['P'] = 1  
>>> y['W'] = (y['F'] > y['A']).astype(int)  
>>> y.loc[y['F'] == y['A'], 'D'] = 1  
>>> y.eval('L = 1*(A>F)', inplace=True)
```

```
>>> y['D'] = y['D'].fillna(0)
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	1	0	0
Carlton	12	11	83	92	1	0	0	1
Gold Coast	17	19	121	60	1	1	0	0
Essendon	9	6	60	121	1	0	0	1
North Melbourne	16	11	107	97	1	1	0	0
Adelaide	14	13	97	107	1	0	0	1
Melbourne	12	8	80	78	1	1	0	0
GW Sydney	10	18	78	80	1	0	0	1
Sydney	18	25	138	53	1	1	0	0
Collingwood	7	11	63	133	1	0	0	1
Port Adelaide	20	13	133	100	1	1	0	0
...	...	...	...	...	...	...	...	...
Gold Coast	7	6	48	168	1	0	0	1
Richmond	8	11	59	94	1	0	0	1
Port Adelaide	13	16	94	59	1	1	0	0
GW Sydney	24	14	158	83	1	1	0	0
Hawthorn	12	11	13	158	1	0	0	1
Brisbane Lions	14	10	94	97	1	0	0	1
Sydney	15	7	97	94	1	1	0	0
Carlton	10	12	72	57	1	1	0	0
Essendon	8	9	57	72	1	0	0	1
West Coast	18	16	124	62	1	1	0	0
Collingwood	9	8	62	124	1	0	0	1

[108 rows x 8 columns]



# Creating the other columns (4)

---

```
>>> y = y.reset_index(['Date','Venue','Round'], drop=True)
```

```
>>> y['P'] = 1  
>>> y['W'] = (y['F'] > y['A']).astype(int)  
>>> y.loc[y['F'] == y['A'], 'D'] = 1  
>>> y.eval('L = 1*(A>F)', inplace=True)
```

```
>>> y['D'] = y['D'].fillna(0)
```

```
>>> y
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	1	0	0
Carlton	12	11	83	92	1	0	0	1
Gold Coast	17	19	121	60	1	1	0	0
Essendon	9	6	60	121	1	0	0	1
North Melbourne	16	11	107	97	1	1	0	0
Adelaide	14	13	97	107	1	0	0	1
Melbourne	12	8	80	78	1	1	0	0
GW Sydney	10	18	78	80	1	0	0	1
Sydney	18	25	138	53	1	1	0	0
Collingwood	7	11	63	133	1	0	0	1
Port Adelaide	20	13	133	100	1	1	0	0
...	...	...	...	...	...	...	...	...
Gold Coast	7	6	48	168	1	0	0	1
Richmond	8	11	59	94	1	0	0	1
Port Adelaide	13	16	94	59	1	1	0	0
GW Sydney	24	14	158	83	1	1	0	0
Hawthorn	12	11	13	158	1	0	0	1
Brisbane Lions	14	10	94	97	1	0	0	1
Sydney	15	7	97	94	1	1	0	0
Carlton	10	12	72	57	1	1	0	0
Essendon	8	9	57	72	1	0	0	1
West Coast	18	16	124	62	1	1	0	0
Collingwood	9	8	62	124	1	0	0	1

[108 rows x 8 columns]

# Creating the other columns (5)

```
>>> y = y.reset_index(['Date','Venue','Round'], drop=True)
```

```
>>> y['P'] = 1  
>>> y['W'] = (y['F'] > y['A']).astype(int)  
>>> y.loc[y['F'] == y['A'], 'D'] = 1  
>>> y.eval('L = 1*(A>F)', inplace=True)
```

```
>>> y['D'] = y['D'].fillna(0).astype(int)
```

Alternatively:

```
>>> y['D'] = 0  
>>> y.loc[y['F'] == y['A'], 'D'] = 1
```

```
>>> y._data.blocks
```

```
(IntBlock: slice(0, 5, 1), 5 x 108, dtype: int64,  
 IntBlock: slice(7, 8, 1), 1 x 108, dtype: int64,  
 IntBlock: slice(5, 6, 1), 1 x 108, dtype: int64,  
 IntBlock: slice(6, 7, 1), 1 x 108, dtype: int64)
```

```
>>> y._data = y._data.consolidate() # _=y.max()
```

```
>>> y._data.blocks
```

```
(IntBlock: slice(0, 8, 1), 8 x 108, dtype: int64,)
```

```
>>> y
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	1	0	0
Carlton	12	11	83	92	1	0	0	1
Gold Coast	17	19	121	60	1	1	0	0
Essendon	9	6	60	121	1	0	0	1
North Melbourne	16	11	107	97	1	1	0	0
Adelaide	14	13	97	107	1	0	0	1
Melbourne	12	8	80	78	1	1	0	0
GW Sydney	10	18	78	80	1	0	0	1
Sydney	18	25	138	53	1	1	0	0
Collingwood	7	11	63	133	1	0	0	1
Port Adelaide	20	13	133	100	1	1	0	0
...	...	...	...	...	...	...	...	...
Gold Coast	7	6	48	168	1	0	0	1
Richmond	8	11	59	94	1	0	0	1
Port Adelaide	13	16	94	59	1	1	0	0
GW Sydney	24	14	158	83	1	1	0	0
Hawthorn	12	11	13	158	1	0	0	1
Brisbane Lions	14	10	94	97	1	0	0	1
Sydney	15	7	97	94	1	1	0	0
Carlton	10	12	72	57	1	1	0	0
Essendon	8	9	57	72	1	0	0	1
West Coast	18	16	124	62	1	1	0	0
Collingwood	9	8	62	124	1	0	0	1

[108 rows x 8 columns]



## -5. Aggregate/subtotal with GroupBy

```
>>> t = y.groupby(by='Team').sum()
```

Team	G	B	F	A	P	W	D	L
Richmond	14	8	92	83	1	1	0	0
Carlton	12	11	83	92	1	0	0	1
Gold Coast	17	19	121	60	1	1	0	0
Essendon	9	6	60	121	1	0	0	1
North Melbourne	16	11	107	97	1	1	0	0
Adelaide	14	13	97	107	1	0	0	1
Melbourne	12	8	80	78	1	1	0	0
GW Sydney	10	18	78	80	1	0	0	1
Sydney	18	25	138	53	1	1	0	0
Collingwood	7	11	63	133	1	0	0	1
Port Adelaide	20	13	133	100	1	1	0	0
...	...	...	...	...	...	...	...	...
Gold Coast	7	6	48	168	1	0	0	1
Richmond	8	11	59	94	1	0	0	1
Port Adelaide	13	16	94	59	1	1	0	0
GW Sydney	24	14	158	83	1	1	0	0
Hawthorn	12	11	13	158	1	0	0	1
Brisbane Lions	14	10	94	97	1	0	0	1
Sydney	15	7	97	94	1	1	0	0
Carlton	10	12	72	57	1	1	0	0
Essendon	8	9	57	72	1	0	0	1
West Coast	18	16	124	62	1	1	0	0
Collingwood	9	8	62	124	1	0	0	1

[108 rows x 8 columns]



# GroupBy isn't necessarily slow

---

```
>>> %timeit pd.concat( [ subDF.sum()  
    for key, subDF in scores.groupby(level=['Venue','Team'])  
    ] )  
1 loop, best of 3: 233 ms per loop
```

```
>>> %timeit scores.groupby(level=['Venue','Team']).sum()  
100 loops, best of 3: 4.22 ms per loop
```



# GroupBy isn't necessarily slow - internals

---

```
>>> %timeit scores.groupby(level=['Venue','Team']).sum()  
100 loops, best of 3: 4.22 ms per loop
```

```
>>> gb = scores.groupby(level=['Venue','Team'])  
>>> gb.grouper.groups['M.C.G.', 'Collingwood'][:3]  
[(Timestamp('1897-06-19 00:00:00'), 'M.C.G.', 'R6', 'Collingwood'),  
 ...  
(Timestamp('2016-04-25 00:00:00'), 'M.C.G.', 'R5', 'Collingwood')]
```

```
>>> idx = gb.grouper.indices['M.C.G.', 'Collingwood']  
array([ 47, 114, 119, 145, 335, 449, 629, 699, 821,  
     826, 919, 968, 985, 1103, 1107, 1199, 1237, 1249,  
     1255, 1367, 1391, 1485, 1535, 1707, ... , 29640, 29660])
```

```
>>> %timeit scores['G'][idx].sum()  
100 loops, best of 3: 2.78 ms per loop
```

```
>>> %timeit scores['G'].values[idx].sum()  
The slowest run took 13.73 times longer than the fastest.  
This could mean that an intermediate result is being cached.  
100000 loops, best of 3: 9.88 µs per loop
```



# Adding points/percentages for each team

---

```
>>> t = y.groupby(by='Team').sum()
```

```
>>> t['PCT'] = 100.0 * t.F / t.A
```

```
>>> t['PTS'] = 4 * t['W'] + 2 * t['D']
```

```
>>> ladder = t.sort_values(['PTS', 'PCT'],  
ascending=False)
```

```
>>> ladder
```

Team	G	B	F	A	P	W	D	L	PCT	PTS
North Melbourne	101	66	672	538	6	6	0	0	124.907063	24
Geelong	97	82	654	380	6	5	0	1	174.736842	20
Sydney	94	81	615	448	6	5	0	1	143.973214	20
Western Bulldogs	77	74	536	344	6	4	0	2	155.813953	16
GW Sydney	98	82	670	476	6	4	0	2	140.756303	16
West Coast	88	81	609	467	6	4	0	2	130.406852	16
Adelaide	102	76	588	564	6	4	0	2	121.985816	16
Hawthorn	82	68	560	610	6	4	0	2	91.803279	16
Melbourne	93	47	605	592	6	3	0	3	102.195946	12
Gold Coast	81	79	565	595	6	3	0	3	94.957983	12
Port Adelaide	78	81	549	612	6	3	0	3	89.705882	12
St Kilda	83	55	553	625	6	2	0	4	88.480000	8
Collingwood	74	57	501	637	6	2	0	4	78.649922	8
Carlton	54	64	388	528	6	2	0	4	73.484848	8
Richmond	70	62	482	646	6	1	0	5	74.613003	4
Brisbane Lions	71	83	509	719	6	1	0	5	70.792768	4
Essendon	52	51	63	586	6	1	0	5	61.945392	4
Fremantle	58	82	130	622	6	0	0	6	69.131833	0



# Final formatting adjustment

Toyota AFL Premiership Season Ladder

SEASON		2016	ROUND	All Rounds	Advanced Options ▾						
Pos	Club	P	W	L	D	F	A	%	Pts		
1	North Melbourne	6	6	0	0	672	538	124.9	24		
2	Geelong Cats	6	5	1	0	664	380	174.7	20		
3	Sydney Swans	6	5	1	0	645	448	144	20		
4	Western Bulldogs	6	4	2	0	536	344	155.8	16		
5	GWS Giants	6	4	2	0	670	476	140.8	16		
6	West Coast Eagles	6	4	2	0	609	467	130.4	16		
7	Adelaide Crows	6	4	2	0	688	564	122	16		
8	Hawthorn	6	4	2	0	560	610	91.8	16		
9	Melbourne	6	3	3	0	605	592	102.2	12		
10	Gold Coast Suns	6	3	3	0	565	595	95	12		
11	Port Adelaide	6	3	3	0	549	612	89.7	12		
12	St Kilda	6	2	4	0	553	625	88.5	8		
13	Collingwood	6	2	4	0	501	637	78.7	8		
14	Carlton	6	2	4	0	388	528	73.5	8		
15	Richmond	6	1	5	0	482	646	74.6	4		
16	Brisbane Lions	6	1	5	0	509	719	70.8	4		
17	Essendon	6	1	5	0	363	586	62	4		
18	Fremantle	6	0	6	0	430	622	69.1	0		

```
>>> pd.options.display.float_format = "%0.1f"
>>> ladder['Pos'] = pd.RangeIndex(1,len(ladder)+1)
>>> ladder
```

Team	G	B	F	A	P	W	D	L	PCT	PTS	Pos
North Melbourne	101	66	672	538	6	6	0	0	124.9	24	1
Geelong	97	82	654	380	6	5	0	1	174.7	20	2
Sydney	94	81	615	448	6	5	0	1	144.0	20	3
Western Bulldogs	77	74	536	344	6	4	0	2	155.8	16	4
GW Sydney	98	82	670	476	6	4	0	2	140.8	16	5
West Coast	88	81	609	467	6	4	0	2	130.4	16	6
Adelaide	102	76	688	564	6	4	0	2	122.0	16	7
Hawthorn	82	68	560	610	6	4	0	2	91.8	16	8
Melbourne	93	47	605	592	6	3	0	3	102.2	12	9
Gold Coast	81	79	665	595	6	3	0	3	95.0	12	10
Port Adelaide	78	81	549	612	6	3	0	3	89.7	12	11
St Kilda	83	55	553	625	6	2	0	4	88.5	8	12
Collingwood	74	57	501	637	6	2	0	4	78.6	8	13
Carlton	54	64	388	528	6	2	0	4	73.5	8	14
Richmond	70	62	482	646	6	1	0	5	74.6	4	15
Brisbane Lions	71	83	509	719	6	1	0	5	70.8	4	16
Essendon	52	51	63	586	6	1	0	5	61.9	4	17
Fremantle	58	82	630	622	6	0	0	6	69.1	0	18

379

# Using the sample code:

```
>>> import pfi
>>> df = pfi.load_data('bg3.txt')
>>> scores = pfi.prepare_game_scores(df)
>>> ladder = pfi.calc_team_ladder(scores, year=2016)
```



```
def calc_team_ladder(scores_df, year=2016):
    """
    DataFrame with championship ladder from round-robin games for the given year.
    Wins, draws and losses are worth 4, 2 and 0 points respectively.
    """
    # Select a subset of the rows
    # df.loc[] matches dates as strings like '20160506' or '2016'.
    if pd.__version__ > '0.18.0':
        # MultiIndex slicing works ok
        scores2 = scores_df.sort_index()
        x = scores2.loc(axis=0)[str(year), :, 'R1':'R9', :]
    else:
        # pandas 0.18.0 has a bug with .loc on MultiIndexes if dates are first level.
        scores2 = scores_df.reorder_levels([1, 2, 3, 0]).sort_index()
        x = scores2.loc(axis=0)[:, 'R1':'R9', :, str(year):str(year)]
        # Don't need to put levels back in order as we are about to drop 3 of them
        x = x.reorder_levels([3, 0, 1, 2]).sort_index()

    # Just keep Team. This does a copy too, avoiding SettingWithCopyWarning
    y = x.reset_index(['Date', 'Venue', 'Round'], drop=True)

    # Add cols with 0/1 for number of games played, won, drawn and lost
    y['P'] = 1
    y['W'] = (y['F'] > y['A']).astype(int)
    y['D'] = 0
    y.loc[y['F'] == y['A'], 'D'] = 1
    y.eval('L = 1*(A>F)', inplace=True)

    # Subtotal by team and then sort by Points/Percentage
    t = y.groupby(level='Team').sum()
    t['PCT'] = 100.0 * t.F / t.A
    t['PTS'] = 4 * t['W'] + 2 * t['D']
    ladder = t.sort_values(['PTS', 'PCT'], ascending=False)

    # Add ladder position (note: assumes no ties!)
    ladder['Pos'] = pd.RangeIndex(1, len(ladder) + 1)

    return ladder
```



---

# Matplotlib

## Tema 11

---



# matplotlib

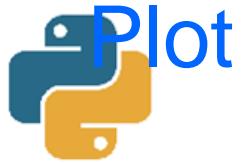
---

A major design limitation is that it strives to emulate MATLAB

More on this in the next lecture

One important function for HW6:

**plot(xvalues, yvalues)**



---

```
import matplotlib.pyplot as plt

xs = [1,2,3,4,5]
ys = [x**2 for x in xs]

plt.plot(xs, ys)
```



no return value?

- We are operating on a “hidden” variable representing the figure.
  - This is a terrible, terrible trick.
  - Its only purpose is to pander to MATLAB users.
  - I’ll show you how this works in the next lecture
-



```
import matplotlib.pyplot as plt  
  
xs = range(-100,100,10)  
x2 = [x**2 for x in xs]  
negx2 = [-x**2 for x in xs]  
  
plt.plot(xs, x2)  
plt.plot(xs, negx2)  
plt.xlabel("x")  
plt.ylabel("y")  
plt.ylim(-2000, 2000)  
plt.axhline(0) # horiz line  
plt.axvline(0) # vert line  
plt.savefig("quad.png")  
plt.show()
```



Incrementally  
modify the figure.

Save your figure to a file

Show it on the screen



```
def myplot(xs, ys, description):
    plt.plot(xs, ys, linewidth=2, color='green', linestyle='-', marker='s', label=description)

def setup_plot():
    plt.xlabel("x")
    plt.ylabel("y")
    plt.axhline(0,linestyle=':',color='red')
    plt.axvline(0,linestyle=':',color='red')

def finish_plot():
    plt.legend()
    plt.show()

setup_plot()
myplot(xs,x2,"x**2")
finish_plot()

setup_plot()
myplot(xs,negx2,"-x**2")
finish_plot()
```

We can group these options into functions as usual, but remember that they are operating on a global, hidden variable

---



# Location of deaths in the 1854 London Cholera Epidemic.

X marks the locations of the water pumps

Dr. John Snow





# Anscombe's Quartet

---

I		II		III		IV	
x	y	x	y	x	y	x	y
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.1	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.1	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

---



## Anscombe's Quartet (2)

---

mean of the x values = 9.0

mean of the y values = 7.5

equation of the least-squared regression line:

$$y = 3 + 0.5x$$

sums of squared errors (about the mean) = 110.0

regression sums of squared errors

(variance accounted for by x) = 27.5

residual sums of squared errors

(about the regression line) = 13.75

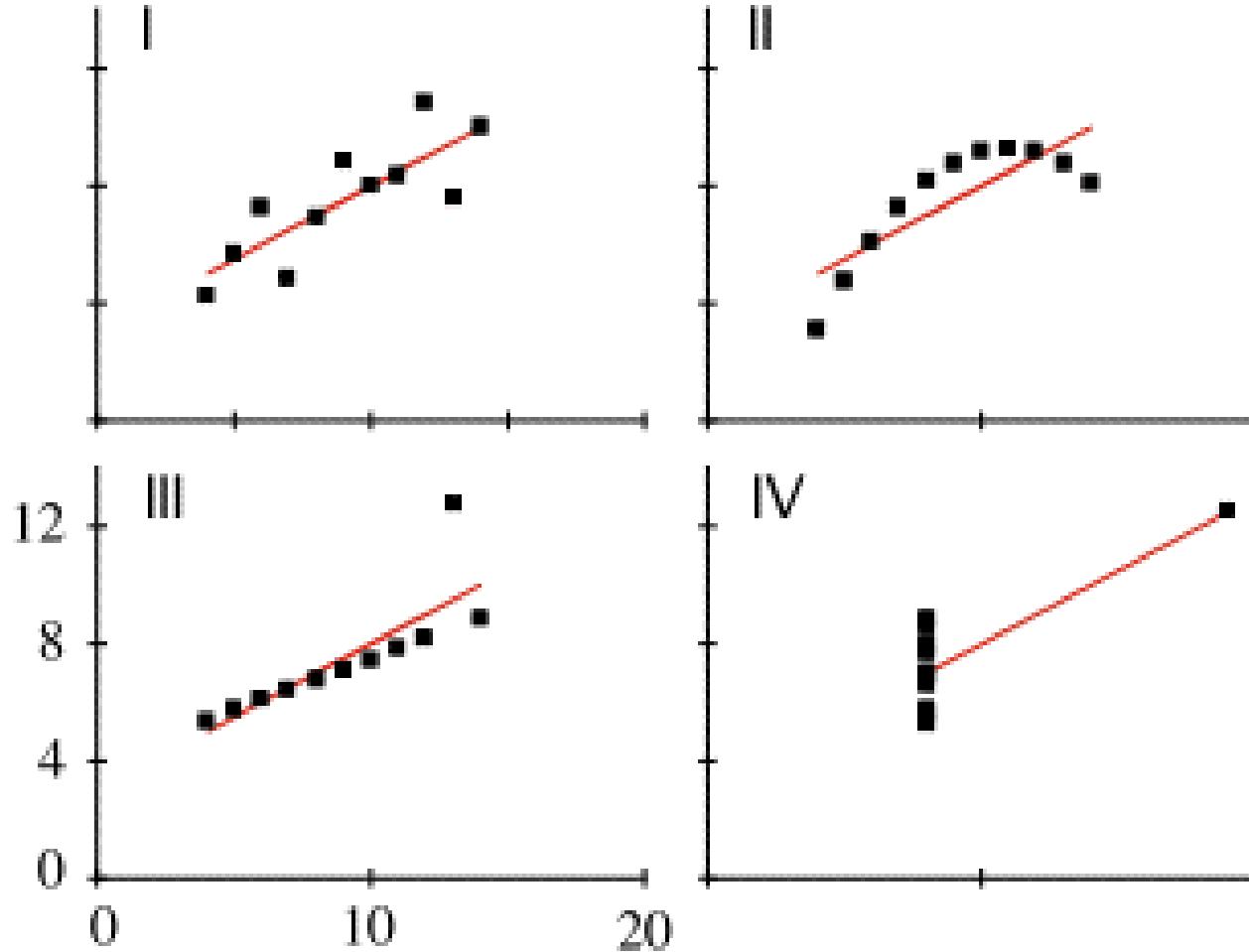
correlation coefficient = 0.82

coefficient of determination = 0.67

---

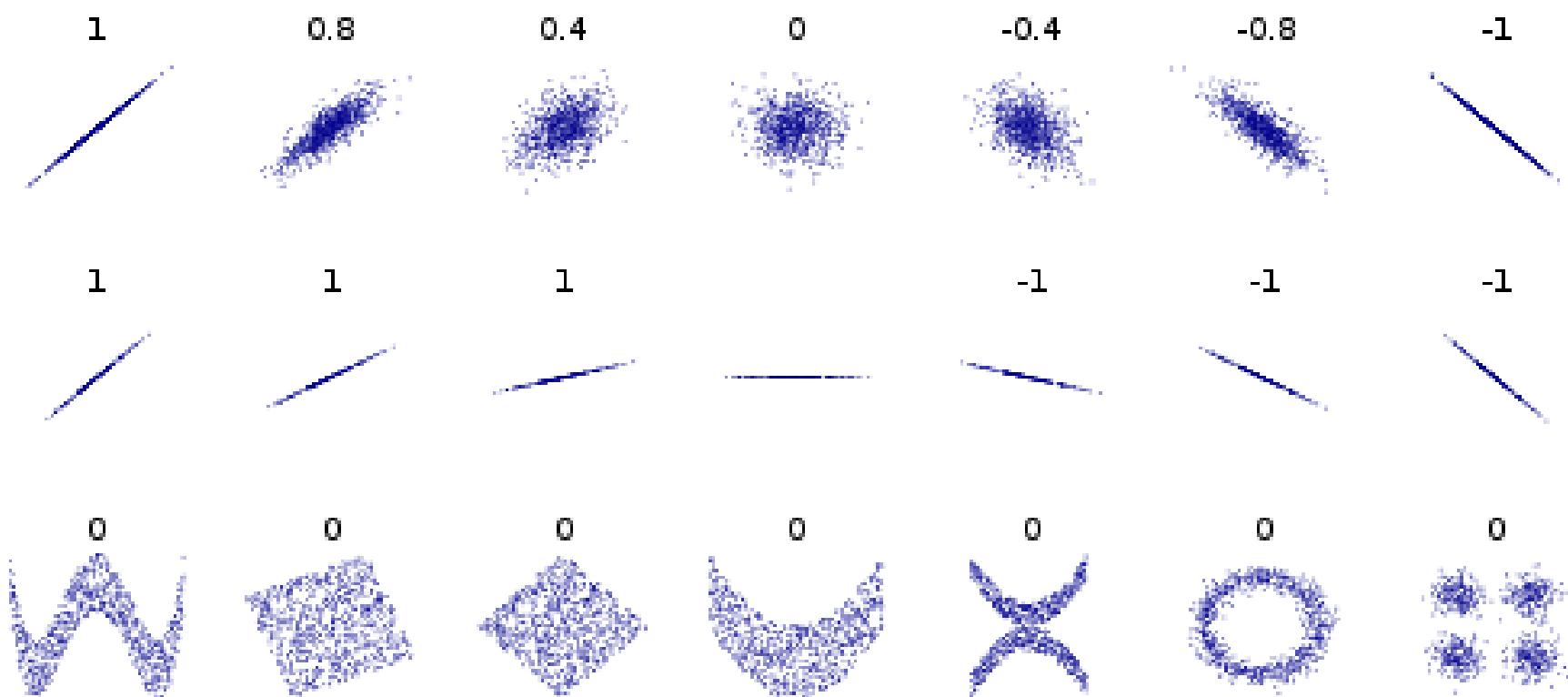


# Anscombe's Quartet (3)





# Another example: Pearson Correlation





# Other reasons?

---

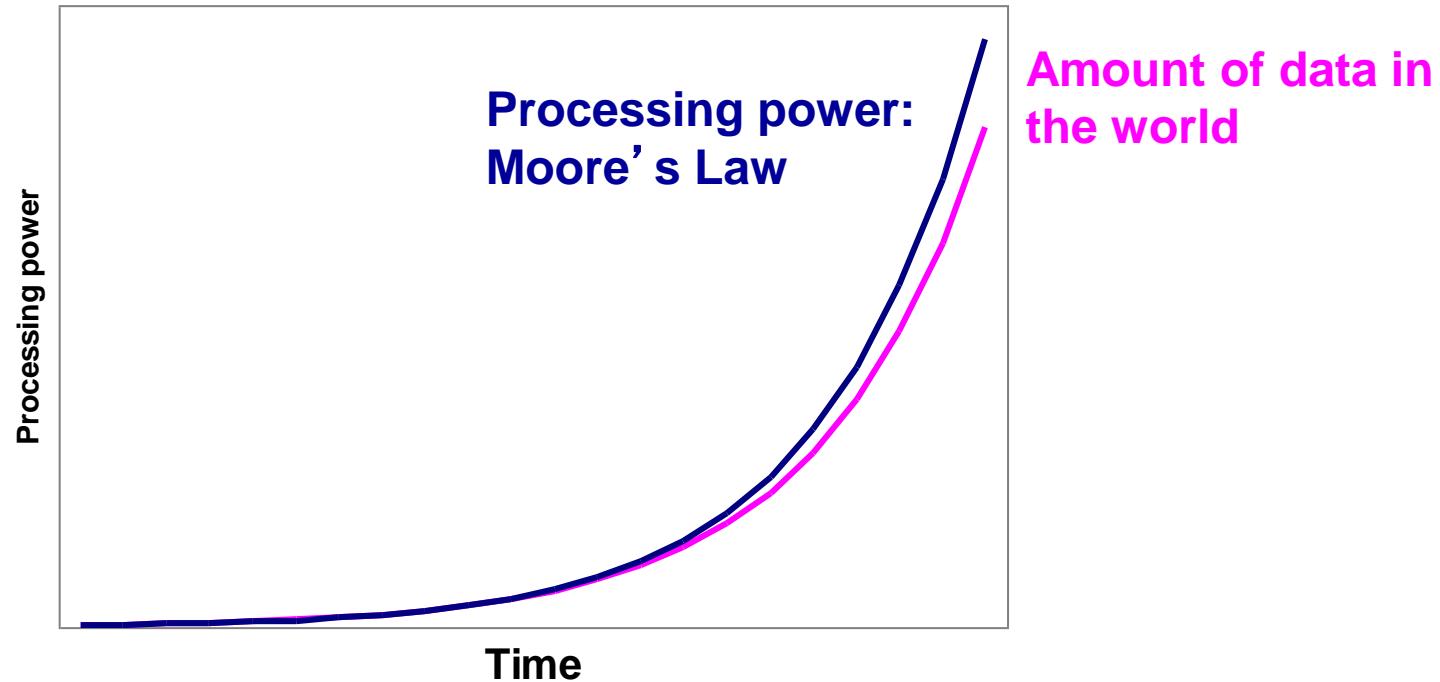
Visualization is the highest bandwidth channel into the human brain [Palmer 99]

The visual cortex is the largest system in the human brain; it's wasteful not to make use of it.

As data volumes grow, visualization becomes a necessity rather than a luxury.

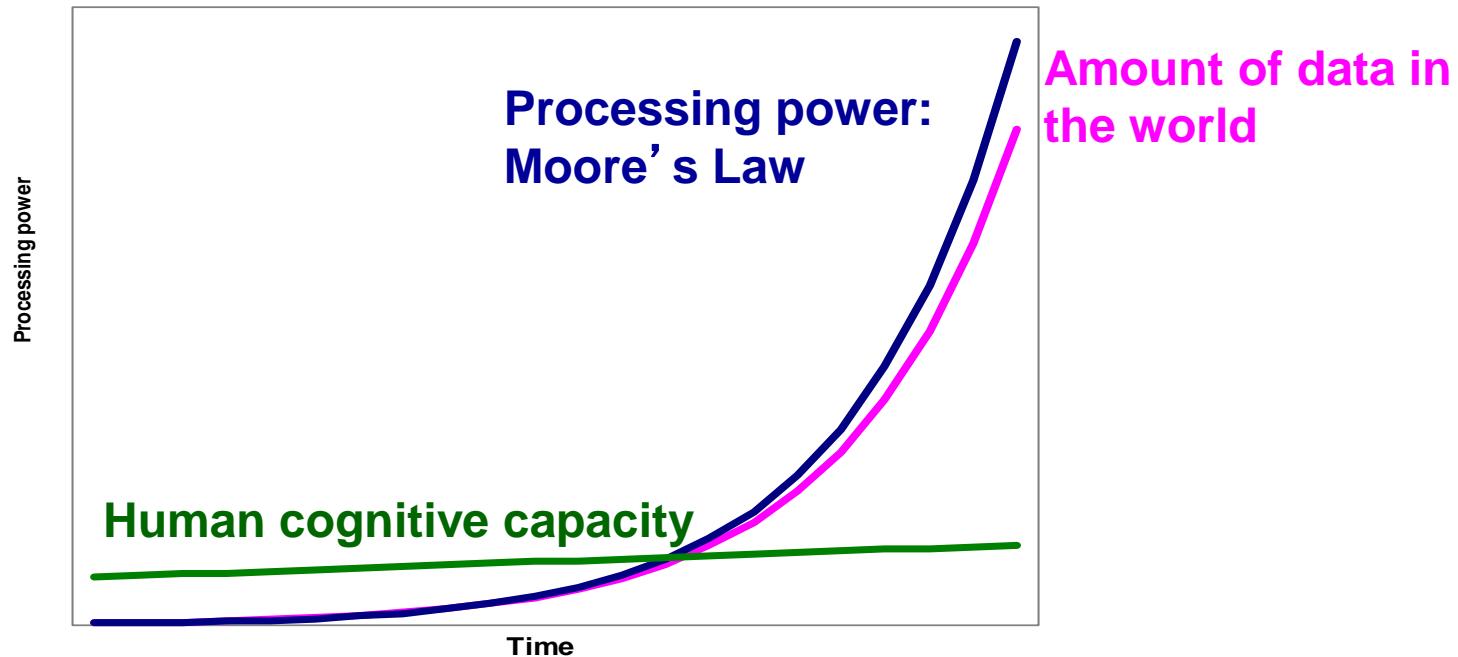
“A picture is worth a thousand words”

# What is the rate-limiting step in data understanding?





# What is the rate-limiting step in data understanding?



Idea adapted from “Less is More” by Bill Buxton (2001)



# What makes a good visualization?

---

Edward Tufte: Minimize the Lie Factor

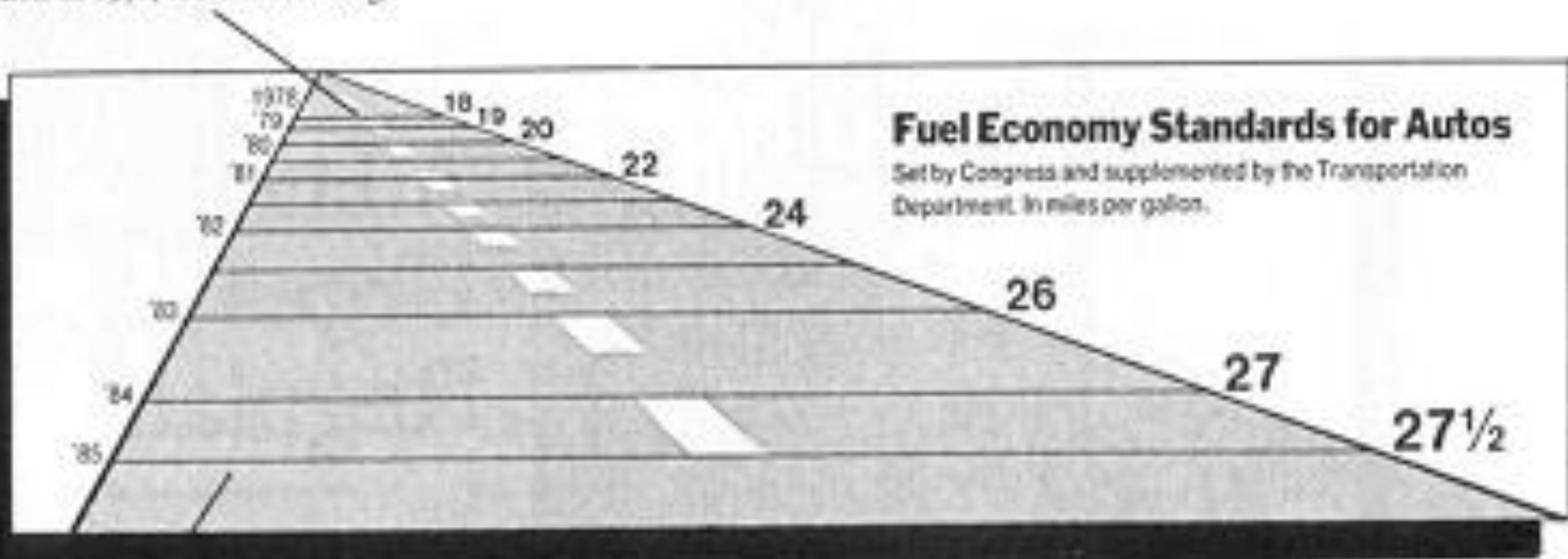


$$\text{Lie Factor} = \frac{\text{Size of effect in the visualization}}{\text{Size of effect in the data}}$$

---

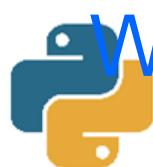
# Example

This line, representing 18 miles per gallon in 1978, is 0.6 inches long.



This line, representing 27.5 miles per gallon in 1985, is 5.3 inches long.

New York Times, August 9, 1978, p. D-2.



# What makes a good visualization?

---

Edward Tufte: Maximize the data-ink ratio

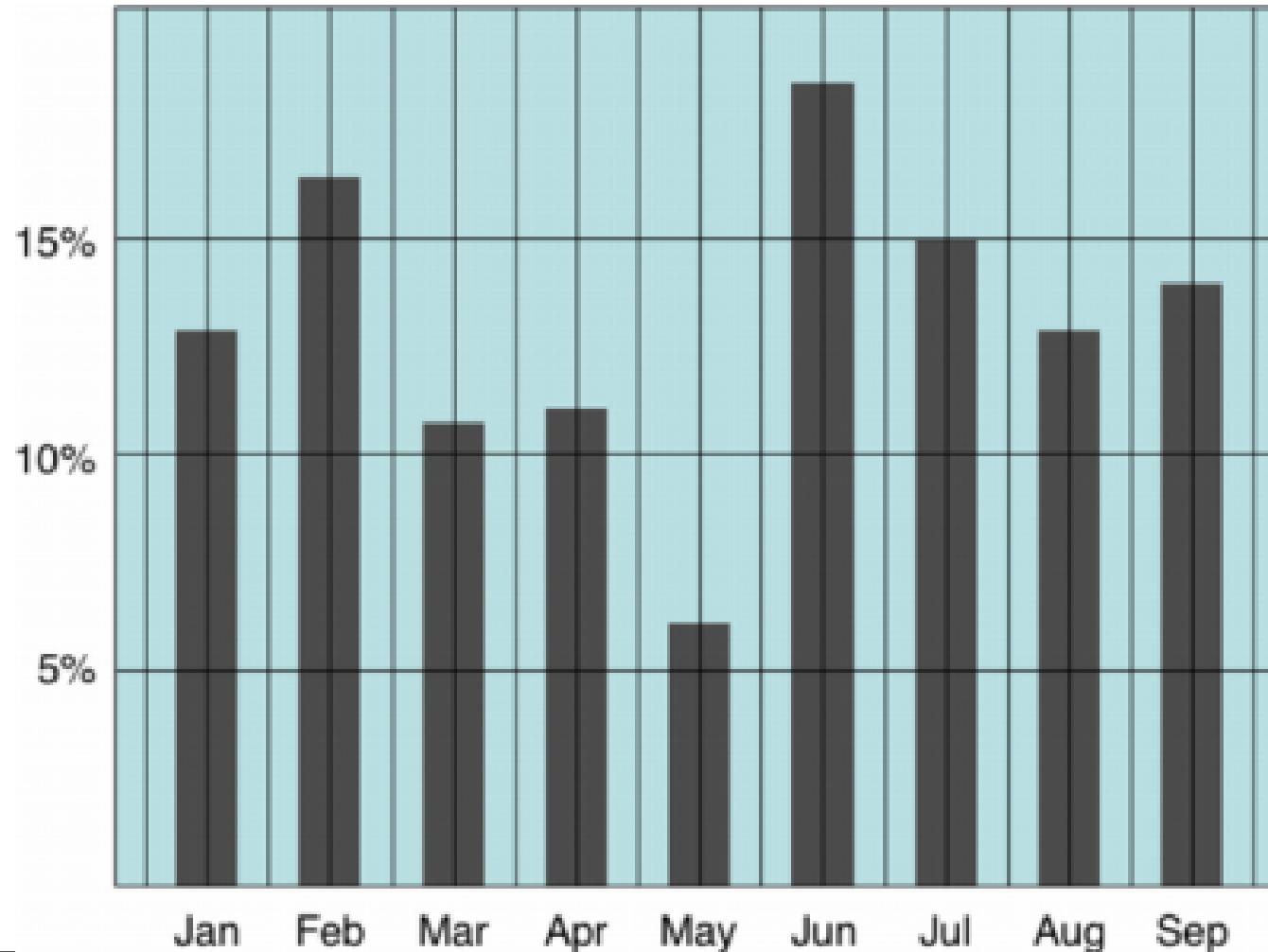


$$\text{Data-ink ratio} = \frac{\text{Data-ink}}{\text{Total ink used to print the graphic}}$$

= proportion of a graphic's ink devoted to the  
non-redundant display of data-information

=  $1.0 - \text{proportion of a graphic that can be erased}$

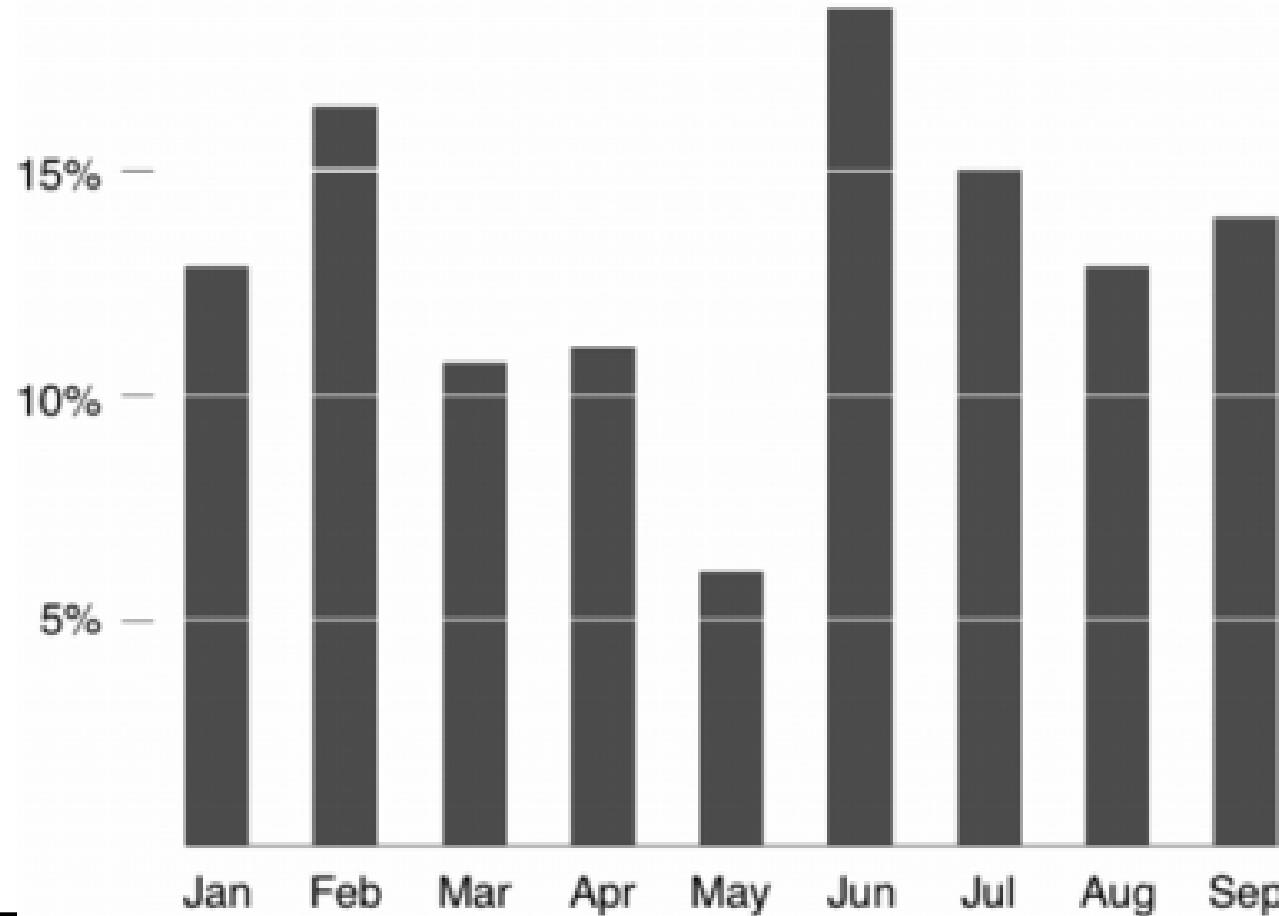
# Example: High or Low Data Ink ratio?





# Example: High or Low Data Ink ratio?

---



# MONSTROUS COSTS

Total House and Senate campaign expenditures,  
in millions





# Bateman et al: The Effects of Visual Embellishment on Comprehension and Memorability of Charts

---

There was no significant difference between plain and image charts for interactive interpretation accuracy (i.e., when the charts were visible).

There was also no significant difference in recall accuracy after a five-minute gap.

After a long-term gap (2-3 weeks), recall of both the chart topic and the details (categories and trend) was significantly better for Holmes charts.

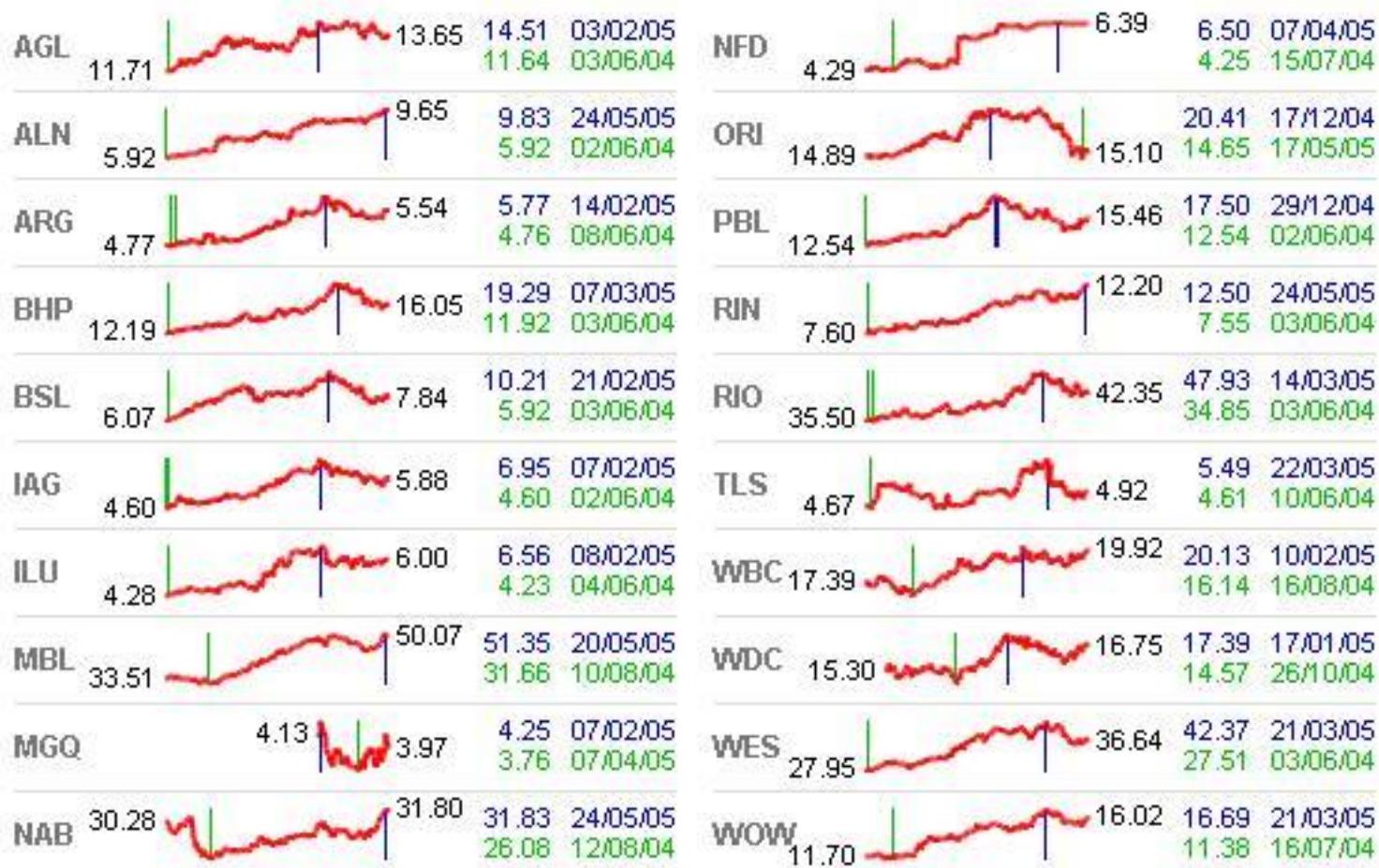
Participants saw value messages in the Holmes charts significantly more often than in the plain charts.

Participants found the Holmes charts more attractive, most enjoyed them, and found that they were easiest and fastest to remember.

---

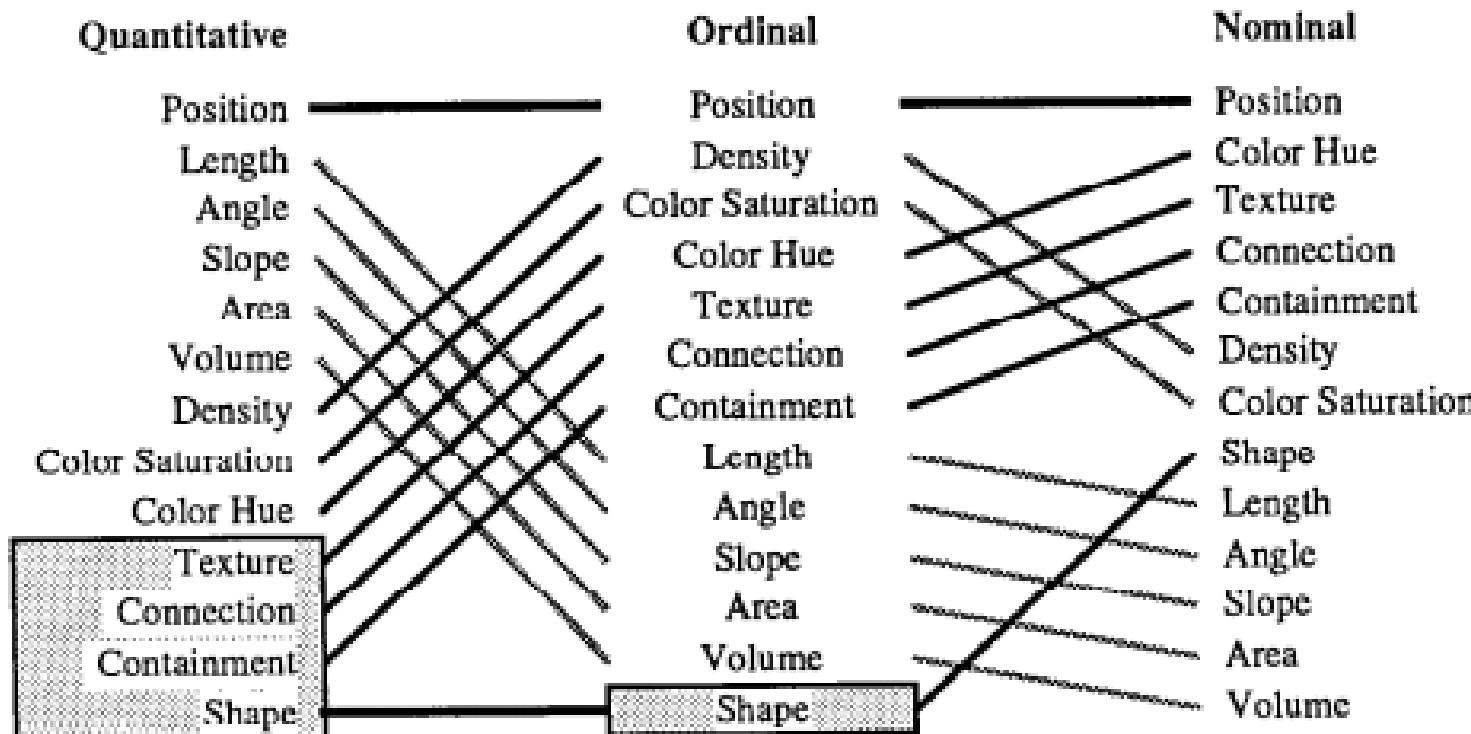
# What makes a good visualization?

Edward Tufte: Small multiples



# What makes a good visualization?

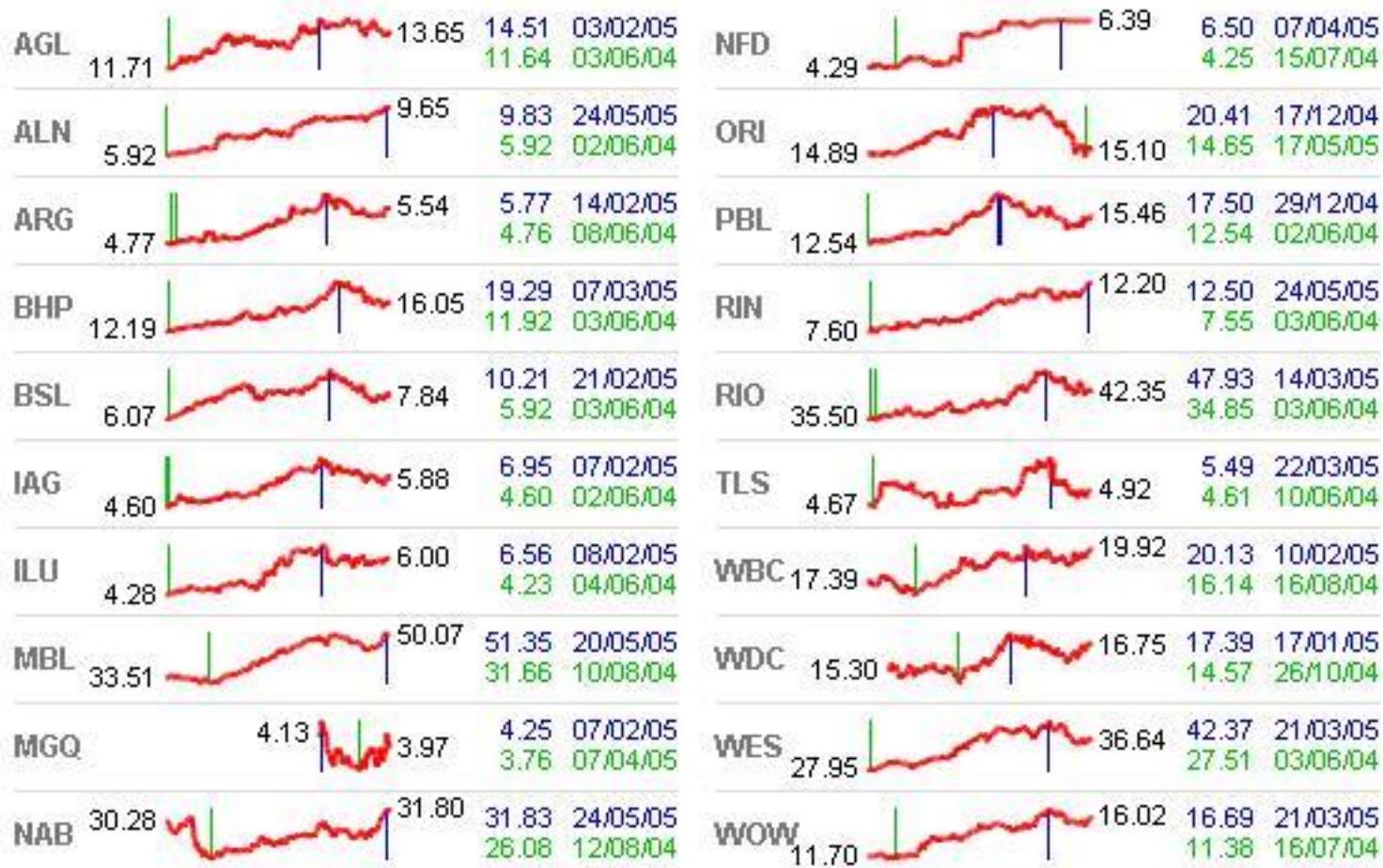
Jock Mackinlay: Use the appropriate visual element for the relationship and data being analyzed



Conjectured rank effectiveness of each visualization method by data type

# What makes a good visualization?

Tufte again: Small multiples

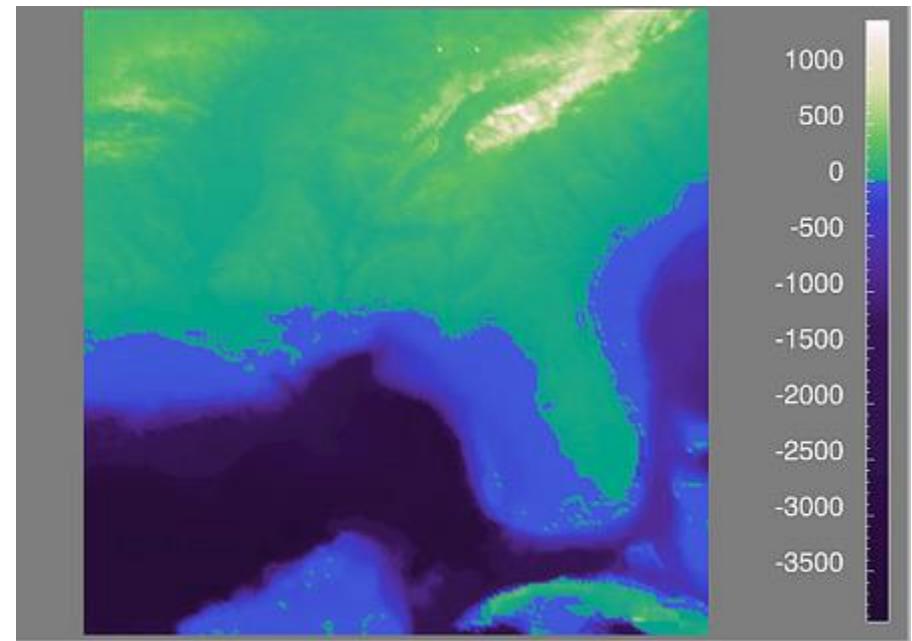
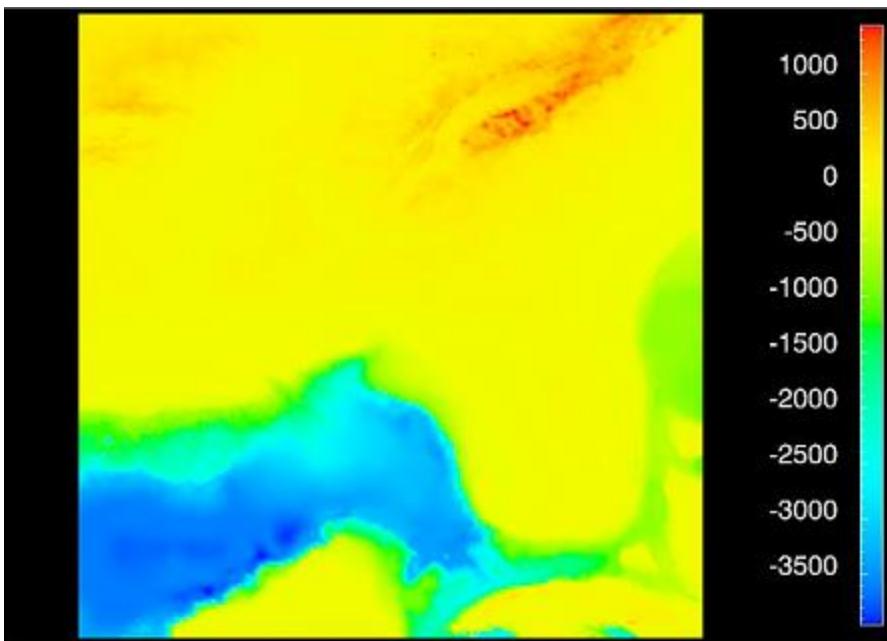




# What makes a good visualization?

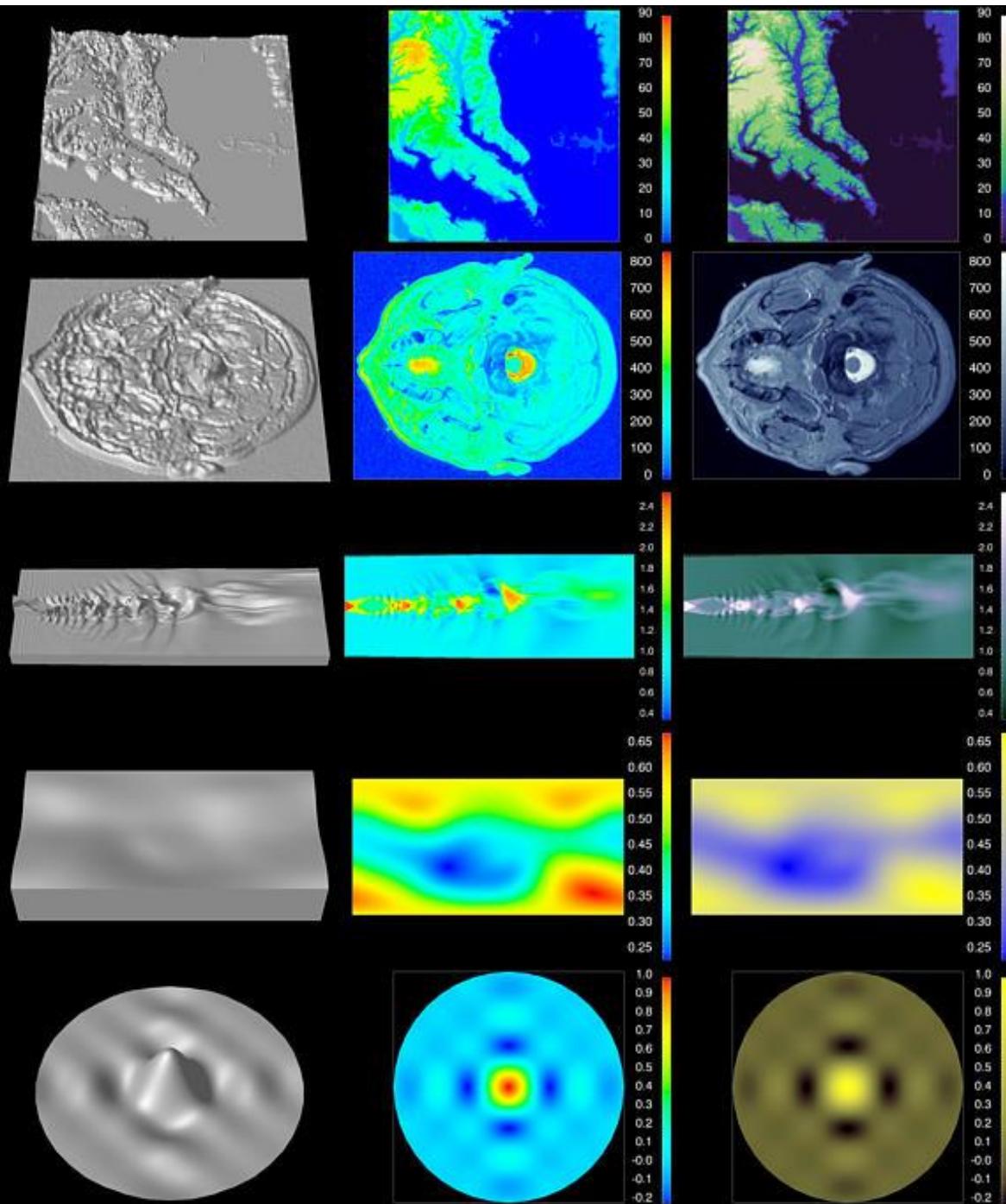
---

## Lloyd Treinish: Color Matters



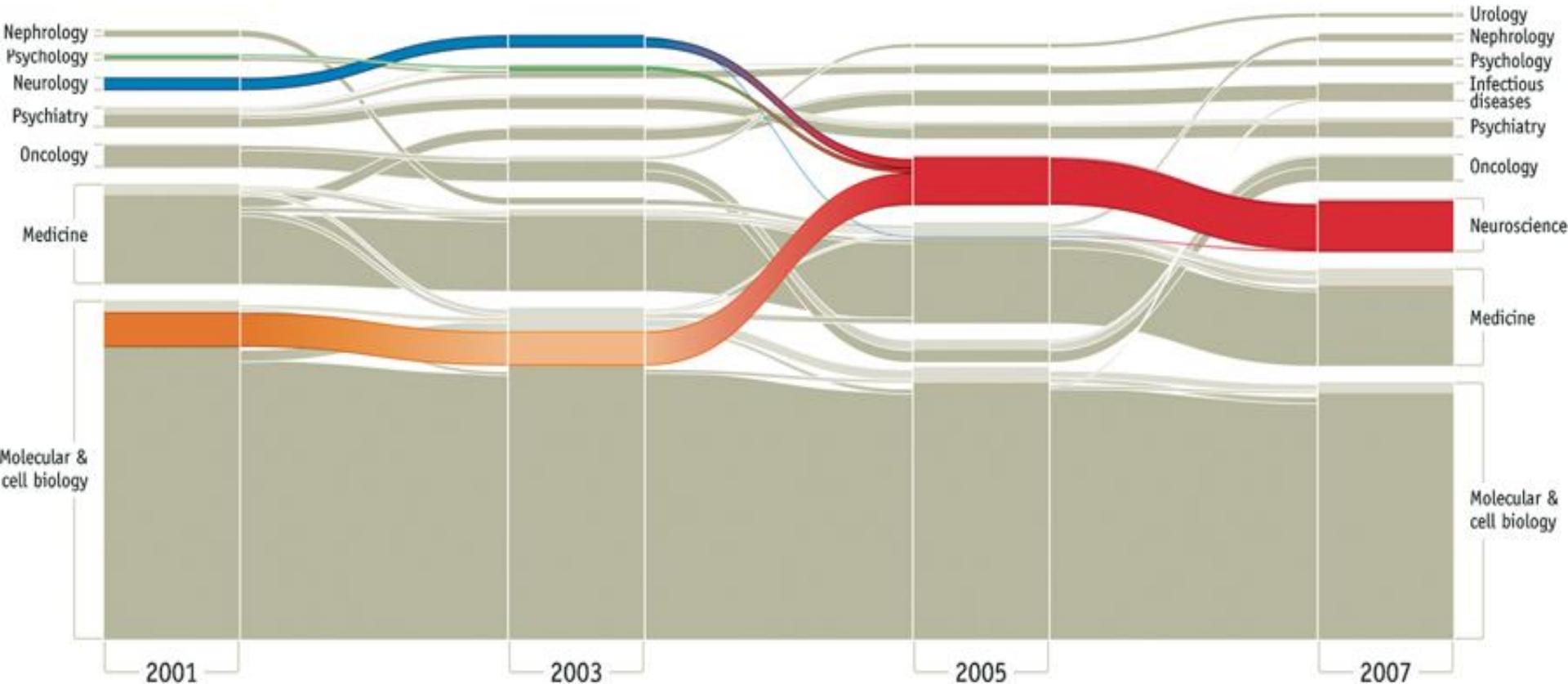
*Lloyd Treinish, IBM Research, <http://www.research.ibm.com/people/l/lloydt/>*

---



*Lloyd Treinish, IBM Research,*  
[http://www.research.  
ibm.com/people/l/llo  
ydt/](http://www.research.ibm.com/people/l/lloyd/)

# A Nice Example



Bergstrom, Rosvall, 2011



---

# Dask

## Tema 10

# Parallel Programming Models

---



# Overview

---

Parallel programming models in common use:

Shared Memory

Threads

Message Passing

Data Parallel

Hybrid

Parallel programming models are abstractions above hardware and memory architectures.

---



## Shared Memory Model

---

Tasks share a common address space, which they read and write asynchronously.

Various mechanisms such as locks / semaphores may be used to control access to the shared memory.

An advantage of this model from the programmer's point of view is that the notion of data "ownership" is lacking, so there is no need to specify explicitly the communication of data between tasks.

Program development can often be simplified.

---



# Disadvantage

---

Is difficult to understand and manage data locality.

Keeping data local to the processor that works on it conserves memory accesses, cache refreshes and bus traffic that occurs when multiple processors use the same data.

Unfortunately, controlling data locality is hard to understand and beyond the control of the average user.

---



# Implementations

---

The native compilers translate user program variables into actual memory addresses, which are global.

Common distributed memory platform implementations does not exist.

A shared memory view of data even though the physical memory of the machine was distributed, impended as virtual shared memory

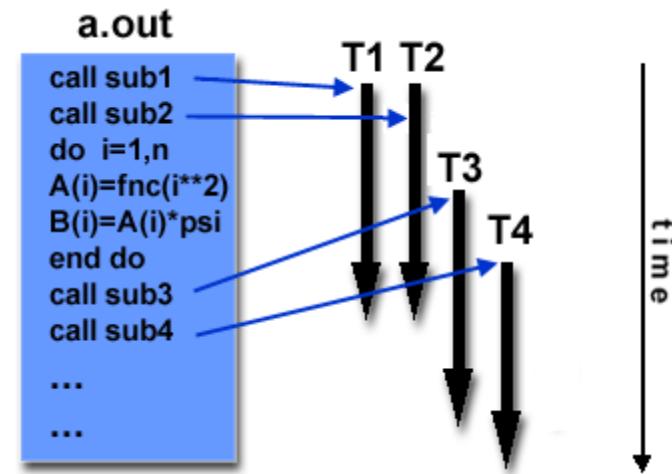
---



# Threads Model

A single process can have multiple, concurrent execution paths.

The main program loads and acquires all of the necessary system and user resources .  
It performs some serial work, and then creates a number of tasks (threads) that run concurrently.





## Threads Cont.

---

The word of a thread can be described as a subroutine within the main program.

All the thread shares the memory space

Each thread has local data.

They save the overhead of replicating the program's resources.

Threads communicate with each other through global memory.

Threads requires synchronization constructs to insure that more than one thread is not updating the same global address at any time.

Threads can come and go, but main thread remains present to provide the necessary shared resources until the application has completed.

---

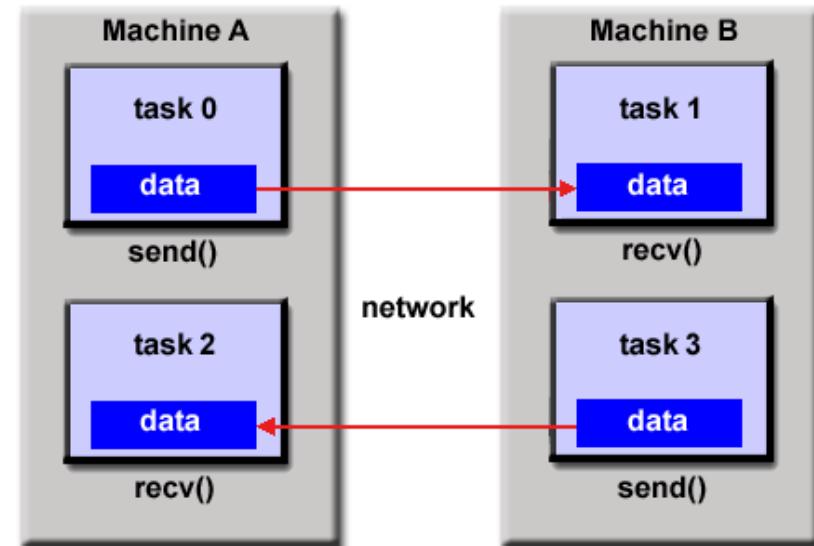


# Message Passing Model

Message Passing Model is used

A set of tasks that use their own local memory during computation.

Multiple tasks can reside on the same physical machine as well across an arbitrary number of machines.





# Message Passing Model

---

Tasks exchange data through communications by sending and receiving messages.

Data transfer usually requires cooperative operations to be performed by each process.

The communication processes may exist on the same machine or different machines

---

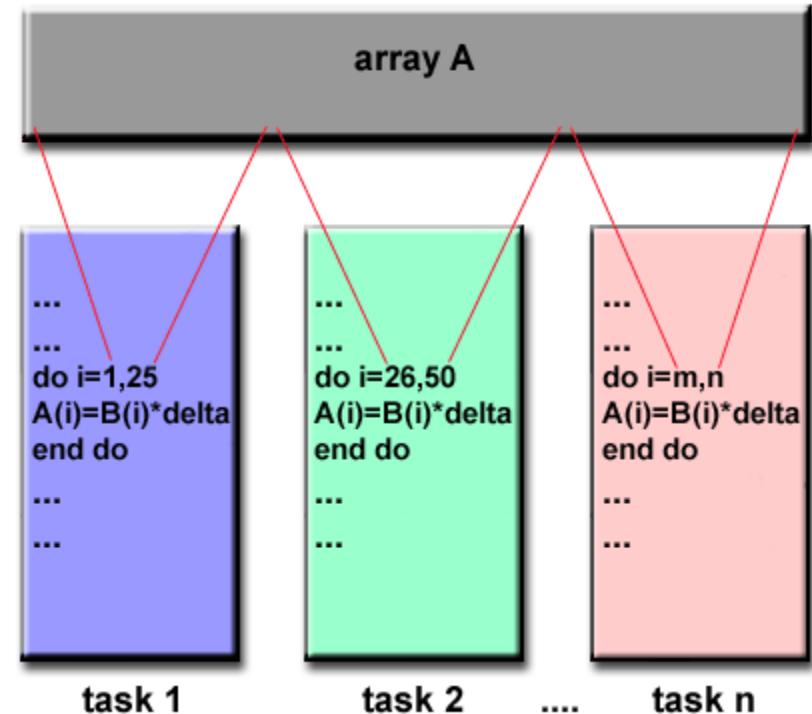


# Data Parallel Model

Most of the parallel work focuses on performing operations on a data set.

The data set is typically organized into a common structure.

A set of tasks work collectively on the same data structure, each task works on a different partition of the same data structure.





## Data Parallel Model Cont.

---

Tasks perform the same operation on their partition of work.

On shared memory architectures, all tasks may have access to the data structure through global memory. On distributed memory architectures the data structure is split up and resides as "chunks" in the local memory of each task.

---



# Designing Parallel Algorithms

---

The programmer is typically responsible for both identifying and actually implementing parallelism. Manually developing parallel codes is a time consuming, complex, error-prone and iterative process.

Currently, The most common type of tool used to automatically parallelize a serial program is a parallelizing compiler or pre-processor.

---



# A parallelizing compiler

---

## Fully Automatic

The compiler analyzes the source code and identifies opportunities for parallelism. The analysis includes identifying inhibitors to parallelism and possibly a cost weighting on whether or not the parallelism would actually improve performance.

Loops (do, for) loops are the most frequent target for automatic parallelization.

## Programmer Directed

Using "compiler directives" or possibly compiler flags, the programmer explicitly tells the compiler how to parallelize the code.

May be able to be used in conjunction with some degree of automatic parallelization also.

---



# Automatic Parallelization Limitations

---

Wrong results may be produced

Performance may actually degrade

Much less flexible than manual parallelization

Limited to a subset (mostly loops) of code

May actually not parallelize code if the analysis suggests there are inhibitors or the code is too complex

---



# The Problem & The Programm

---

Determine whether or not the problem is one that can actually be parallelized.

Identify the program's **hotspots**:

- Know where most of the real work is being done.

- Profilers and performance analysis tools can help here

- Focus on parallelizing the hotspots and ignore those sections of the program that account for little CPU usage.

Identify **bottlenecks** in the program

- Identify areas where the program is slow, or bounded.

- May be possible to restructure the program or use a different algorithm to reduce or eliminate unnecessary slow areas

Identify inhibitors to parallelism. One common class of inhibitor is *data dependence*, as demonstrated by the Fibonacci sequence.

Investigate other algorithms if possible. This may be the single most important consideration when designing a parallel application.

---



# Partitioning

---

Break the problem into discrete "chunks" of work that can be distributed to multiple tasks.

domain decomposition

functional decomposition.



# Domain Decomposition

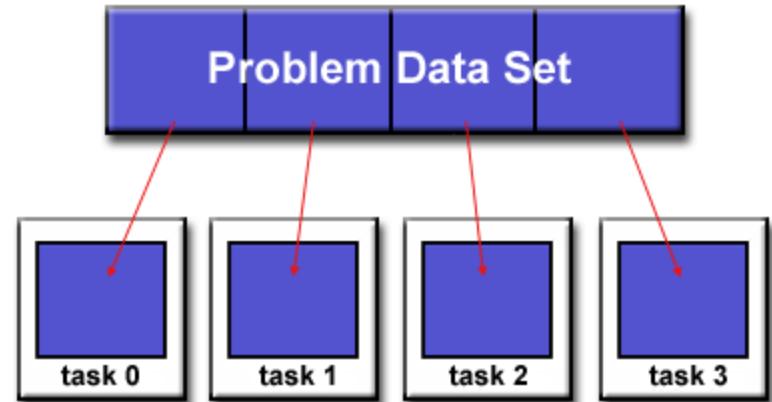
---

The data associated with a problem is decomposed.

Each parallel task then works on a portion of the data.

This partition could be done in different ways.

Row, Columns, Blocks, Cyclic,  
etc.

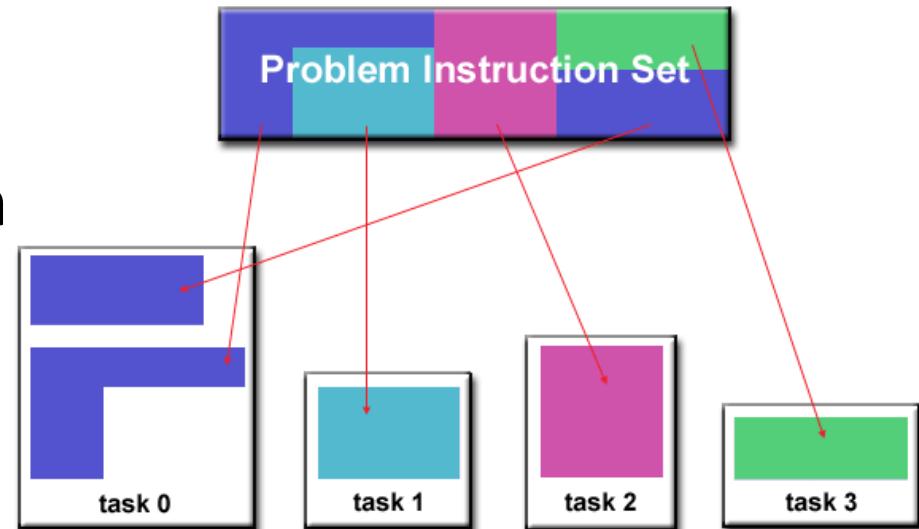




# Functional Decomposition

---

The problem is decomposed according to the work that must be done. Each task then performs a portion of the overall work.





# Communications

---

Cost of communications

Latency vs. Bandwidth

Visibility of communications

Synchronous vs. asynchronous communications

Scope of communications

*Point-to-point*

*Collective*

Efficiency of communications

Overhead and Complexity

---



# Synchronization

---

Barrier

Lock / semaphore

Synchronous communication operations

---



# Data Dependencies

---

A dependence exists between program statements when the order of statement execution affects the results of the program.

A data dependence results from multiple use of the same location(s) in storage by different tasks.

Dependencies are important to parallel programming because they are one of the primary inhibitors to parallelism.

---



# Load Balancing

---

Load balancing refers to the practice of distributing work among tasks so that all tasks are kept busy all of the time. It can be considered a minimization of task idle time. Load balancing is important to parallel programs for performance reasons. For example, if all tasks are subject to a barrier synchronization point, the slowest task will determine the overall performance.

---



# Dask API

---

What is unique about Dask:

- allow to work with larger datasets making it possible to parallelize computation (e.g. "simple" sorting and “aggregating” functions would otherwise spill on persistent memory).
  - it simplifies the cost of using more complex infrastructure.
  - it is easy to learn for data scientists with a background in the Python (similar syntax) and flexible.
-



# Dask API

---

- Dask is fully implemented in Python and natively scales NumPy, Pandas, and scikit-learn.
  - Dask can be used effectively to work with both medium datasets on a single machine and large datasets on a cluster.
  - Dask can be used as a general framework for parallelizing most Python objects.
  - Dask has a very low configuration and maintenance overhead.
-



# Motivation

---

Dataset type	Size range	Fits in RAM?	Fits on local disk?
Small dataset	Less than 2-4 GB	Yes	Yes
Medium dataset	Less than 2 TB	No	Yes
Large dataset	Greater than 2 TB	No	No

---

Adapted from Data Science with Dask



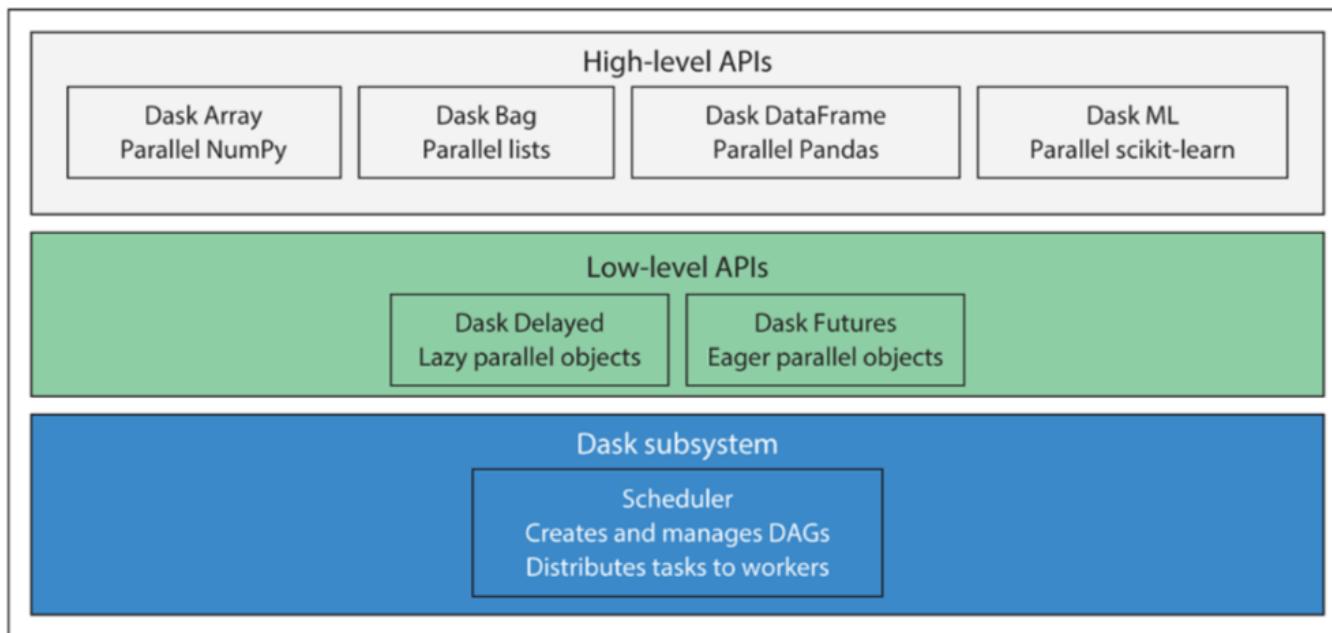
# Dask API <cont>

---

- not of great help for small size datasets: It generates greater overheads. Complex operations can be done without spilling to disk and slowing down process.
  - very useful for medium size dataset: it allows to work with medium size in local machine. Difficult to take advantage of parallelism within Pandas (no sharing work between processes on multicore systems).
  - essential for large datasets: Pandas, NumPy, and scikit-learn are not suitable at all for datasets of this size, because they were not inherently built to operate on distributed datasets.
-



# Dask API <cont>



Adapted from Data Science with Dask



# Directed Acyclical Graph

---

A graph is a representation of a set of objects that have a relationship with one another. It is used to representing a wide variety of information.

A graph is consisted by:

- node: a function, an object or an action
- line: symbolize the relationship among nodes

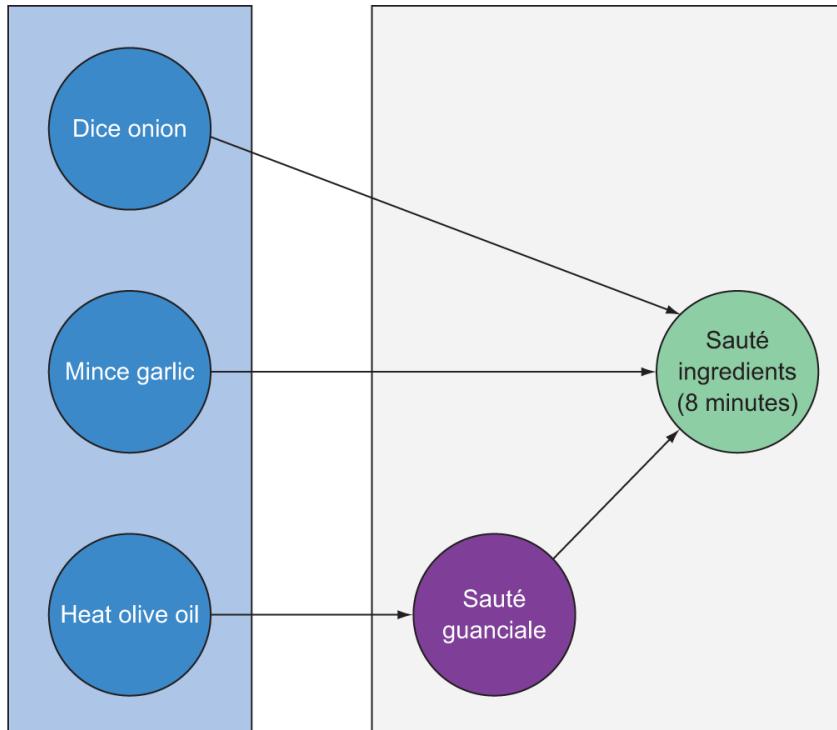
In a directed acyclical graph there is one logical way to traverse the graph. No node is visited twice.

In a cyclical graph: exist a feedback loop that allow to revisit and ~~repeat the actions within the same node~~.

---



# Directed Acyclical Graph <cont>

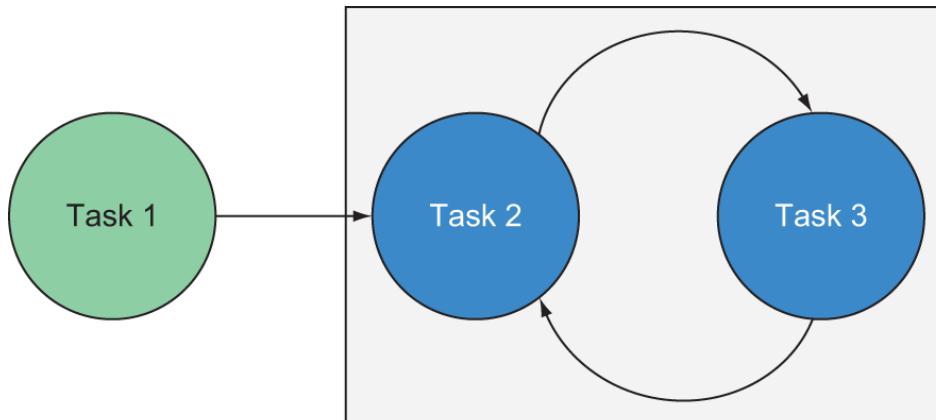


No dependencies.  
These tasks can be started  
in any order.

These tasks can only be  
started when all nodes  
connected to them have been  
completed.



# Directed Acyclical Graph <cont>



Task 2 and Task 3 are connected to each other in an infinite feedback loop. There is no logical termination point in this graph.



# Computational Resources

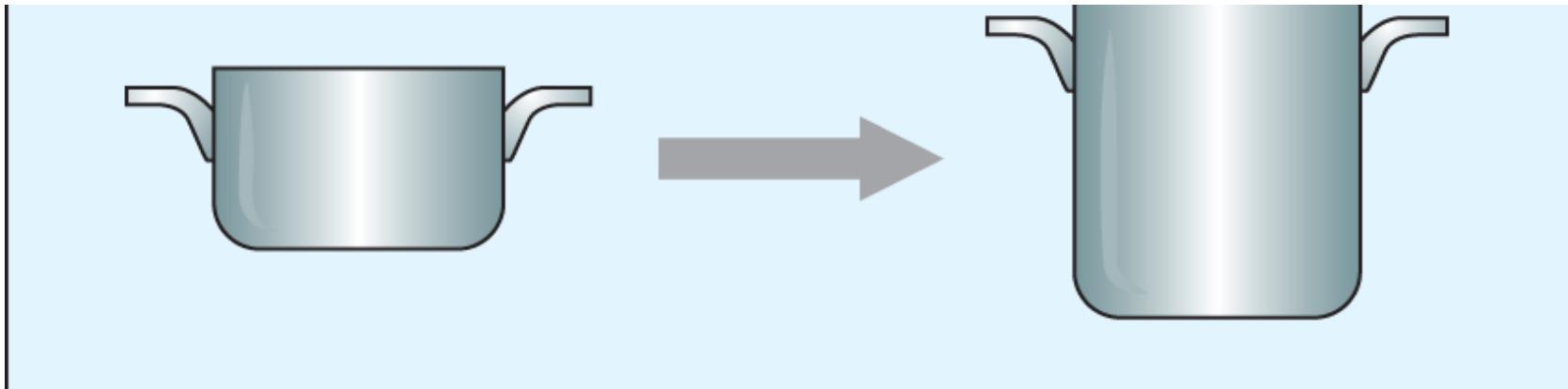
---

How to handle computational resources? As the problem we solve requires more resources we have two options:

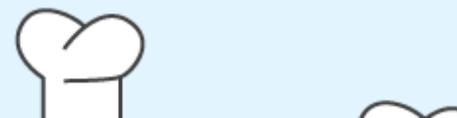
- scale up: increase size of the available resource: invest in more efficient technology. cons diminishing return.
  - scale out: add other resources (dask's main idea). Invest in more cheap resources. cons distribute workload.
-



# Computational Resources <cont>



Scale out



Adapted from Data Science with Dask



# Computational Resources <cont>

---

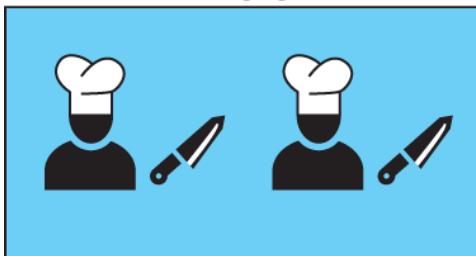
As we approach greater number of "work to be completed", some resources might be not fully exploited. This phenomenon is called concurrency.

For instance some might be idling because of insufficient shared resources (i.e. resource starvation). Schedulers handle this issue by making sure to provide enough resources to each worker.

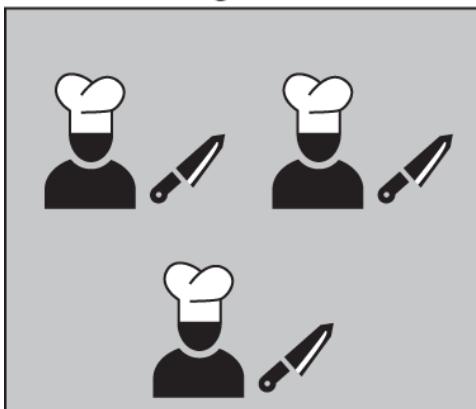


# Computational Resources <cont>

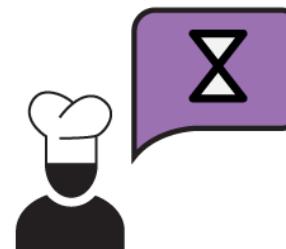
Mincing garlic



Dicing onions



Shared resources



This cook must wait and remain idle until either a knife becomes available or a new task that doesn't require a knife is available. This is an example of a resource-starved worker.

Adapted from Data Science with Dask



# Computational Resources <cont>

---

In case of a failure, Dask reaches a node and repeats the action without disturbing the rest of the process. There are two types of failures:

- work failures: a worker leaves, and you know that you must assign another one to their task. This might potentially slow down the execution, however it won't affect previous work (aka data loss).
  - data loss: some accident happens, and you have to start from the beginning. The scheduler stops and restarts from the beginning the whole process.
-



# Dask Review

---

- Dask can be used to scale popular Python libraries such as Pandas and NumPy allowing to analyze dataset with greater size (>8GB).
  - Dask uses directed acyclical graph to coordinate execution of parallelized code across processors.
  - Upstream actions are completed before downstream nodes.
  - Scaling out (i.e. add workers) can improve performances of complex workloads, however, create overhead that can reduces gains.
  - In case of failure, the step to reach a node can be repeated from the beginning without disturbing the rest of the process.
-



# Task scheduling

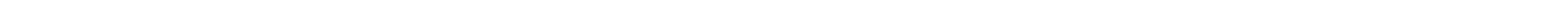
- 
- Dask performs a so called lazy computation. Until you run the method `.compute()`, Dask only splits the process into smaller logical pieces.
  - Even though the process is defined, the number of resources assigned and the place where the result will be stored are not assigned because the scheduler assigns them dynamically. This allow to recover from worker failure.



# Task scheduling <cont>

---

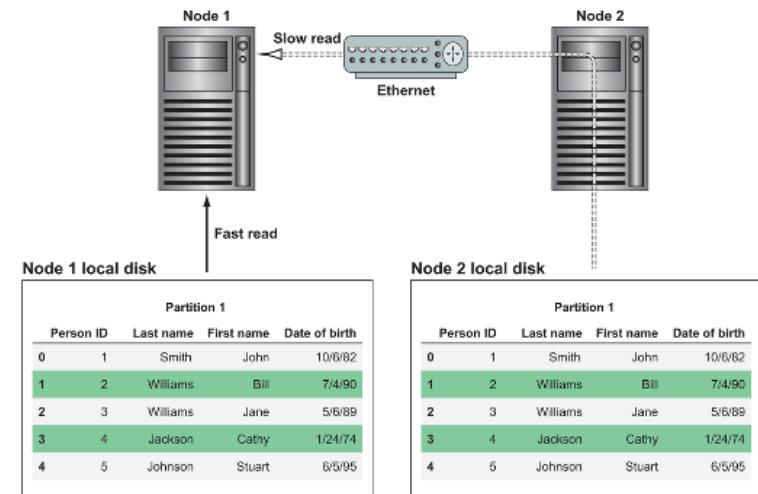
- Dask uses a central scheduler to orchestrate the work. It splits the workload among different servers which they will unlikely be perfectly balanced with respect to load, power and data access. Due to these conditions, scheduler needs to promptly react to avoid bottlenecks that will affect overall runtime.





# Task scheduling <cont>

- For best performance, a Dask cluster should use a distributed file system (S3, HDFS) as a data storage. Assuming there are two nodes like in the image below and data are stored in one. In order to perform computation in the other node we have to move the data from one to the other creating an overhead proportional to the size of the data. The remedy is to split data minimizing the number of data to broadcast across different local machines.





# Dask Review

---

- Dask can be used to scale popular Python libraries such as Pandas and NumPy allowing to analyze dataset with greater size (>8GB).
  - Dask uses directed acyclical graph to coordinate execution of parallelized code across processors.
  - Upstream actions are completed before downstream nodes.
  - Scaling out (i.e. add workers) can improve performances of complex workloads, however, create overhead that can reduces gains.
  - In case of failure, the step to reach a node can be repeated from the beginning without disturbing the rest of the process.
-



# Dask Limitations

---

- Dask dataframe are immutable. Functions such as pop and insert are not supported.
  - Dask does not allow for functions with a lot of data shuffling like stack/unstack and melt.
  - Do major filter and preprocessing in Dask and then dump the final dataset into Pandas.
  - Join, merge, groupby, and rolling are supported but expensive due to shuffling.
  - Do major filter and preprocessing in Dask and then dump the final dataset into Pandas or limit operations only on index which can be pre-sorted.
-



# American “on-time performance” flight data

Monthly zipped csv files

Each file has  
450,000 rows x 109 cols

220MB unzipped  
22MB zipped  
12MB with LZMA (.xz)

[https://www.transtats.bts.gov/acTableInfo.asp?Table\\_ID=236](https://www.transtats.bts.gov/acTableInfo.asp?Table_ID=236)

United States Department of Transportation

Ask a

## Bureau of Transportation Statistics

Explore Topics and Geography      Browse Statistical Products and Data      Learn About BTS and Our Work

**On-Time : On-Time Performance**

Database Profile   Data Tables   Table Contents

Property	Description
Name	On-Time Performance
Description	This table contains on-time arrival data for non-stop domestic flights by major air carriers, and provides such additional items as departure and arrival delays, origin and destination airports, flight numbers, scheduled and actual departure and arrival times, cancelled or diverted flights, taxi-out and taxi-in times, air time, and non-stop distance.
Records	172,695,702
Fields	109
First Year	1987
Last Year	2017
Frequency	Monthly
Latest Available Data	January, 2017
Terms	Definitions
<b>Actual Arrival Times</b>	Gate arrival time is the instance when the pilot sets the aircraft parking brake after arriving at the airport gate or passenger unloading area. If the parking brake is not set, record the time for the opening of the passenger door. Also, carriers using a Docking Guidance System (DGS) may record the official gate-arrival time when the aircraft is stopped at the appropriate parking mark.



## CONGESTION IN THE SKY Visualizing Domestic Airline Traffic with SAS® Software

Rick Wicklin, SAS Institute  
Robert Allison, SAS Institute

### THE DATA

Twenty years of data (120 million observations) on commercial domestic flights in the United States.

#### Variables

- Dates: day of week, date, month, year
- Arrival and departure times: actual and scheduled
- Flight times: actual and scheduled
- Origin and destination: airport code, latitude, longitude
- Carrier: American, Aloha Air, ..., United, US Air

Data are from the Research and Innovative Technology Administration (RITA) which coordinates the U.S. Department of Transportation's travel data programs.

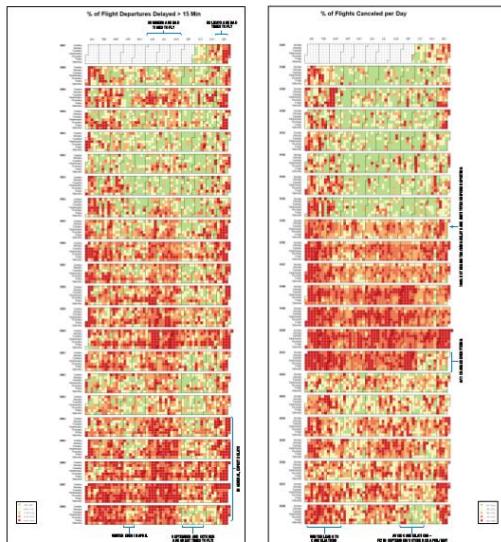
### GOALS

- Summarize data by time periods, airport, and carrier
- Temporal effects
  - Are some time periods more prone to delays than others?
- Relationships between delays and
  - Seasonal factors: month, day of week, holidays
  - Weather factors: blizzards and severe weather
  - Delay factors: time of day, day of week
- Spatial effects
  - Are some airports more prone to delays than others?
  - Are there differences between flying into an airport and flying out?
- Carrier effects
  - Are some carriers more prone to delays than others?

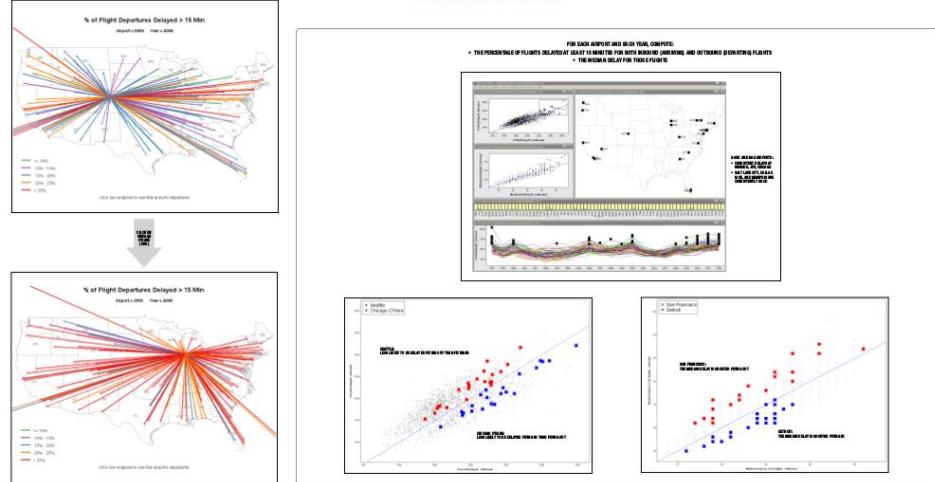
### LESSONS LEARNED: TIPS FOR TRAVELERS

- Avoid flying during holidays and summer
- Fly in April, May, and September
- Watch the weather
- Avoid airports (Newark, JFK, Chicago,...) with consistent delays
- Use carriers (Aloha, Hawaiian, Southwest,...) with superior on-time performance
- Fly early in the day
- Avoid flights that depart between 5 and 7 p.m.

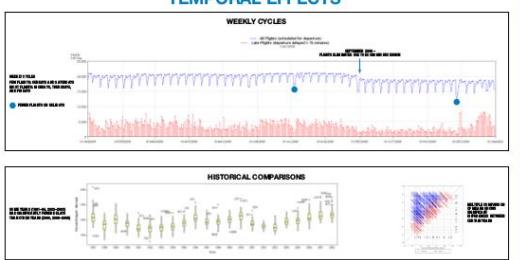
### OVERVIEW



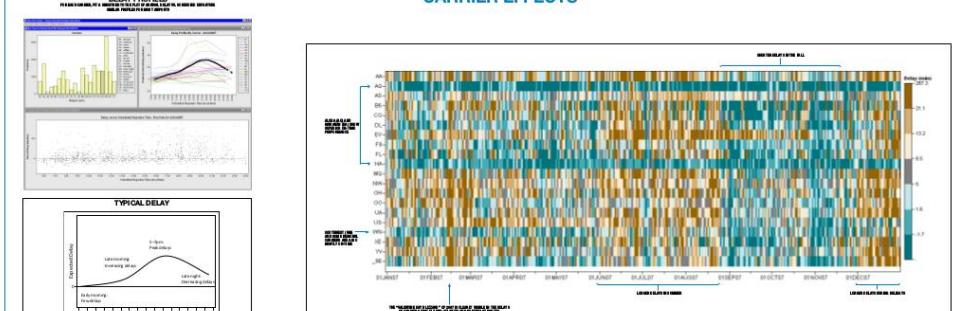
### SPATIAL EFFECTS



### TEMPORAL EFFECTS

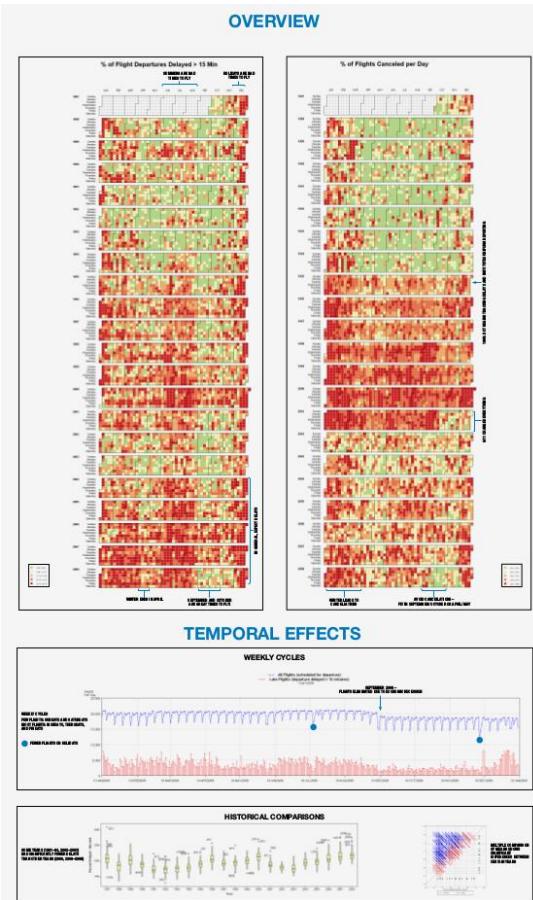


### CARRIER EFFECTS





# Our analysis goals...



Calculate % of flights  
cancelled per day

- by date / date range
- by origin / destination
- by carrier
- by state

451

<http://stat-computing.org/dataexpo/2009>



```
df = pd.read_csv('flights-2016-01.xz', nrows=4, dialect="excel")
```

```
> df.T
```

	2016	2016	2016	2016
Year	2016	2016	2016	2016
Quarter	1	1	1	1
Month	1	1	1	1
DayofMonth	6	7	8	9
DayOfWeek	3	4	5	6
FlightDate	2016-01-06	2016-01-07	2016-01-08	2016-01-09
UniqueCarrier	AA	AA	AA	AA
AirlineID	19805	19805	19805	19805
Carrier	AA	AA	AA	AA
TailNum	N4YBAA	N434AA	N541AA	N489AA
FlightNum	43	43	43	43
...				
Origin	DFW	DFW	DFW	DFW
...				
Dest	DTW	DTW	DTW	DTW
...				
CRSDepTime	1100	1100	1100	1100
DepTime	1057	1056	1055	1102
DepDelay	-3	-4	-5	2
...				
TaxiOut	15	14	21	13
WheelsOff	1112	1110	1116	1115
WheelsOn	1424	1416	1431	1424
TaxiIn	8	10	14	9
CRSArrTime	1438	1438	1438	1438
ArrTime	1432	1426	1445	1433
ArrDelay	-6	-12	7	-5
...				
Cancelled	0	0	0	0
CancellationCode	NaN	NaN	NaN	NaN
Diverted	0	0	0	0
CRSElapsedTime	158	158	158	158
ActualElapsedTime	155	150	170	151
AirTime	132	126	135	129
Flights	1	1	1	1
Distance	986	986	986	986
...				
[110 rows x 4 columns]				



# Pandas DataFrames don't scale well

---

```
> df = pd.read_csv('flights-2016-01.xz',
                    dialect="excel")
```

DtypeWarning: Columns (77) have mixed types.  
Specify dtype option on import or set low\_memory=False.

CPU times: user 5.48 s, sys: 1.36 s, total: 6.83 s

Wall time: 6.83 s

```
> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 445827 entries, 0 to 445826
Columns: 110 entries, Year to Unnamed: 109
dtypes: float64(71), int64(21), object(18)
memory usage: 374.2+ MB
```

```
> df.memory_usage(deep=True).sum() / 2**20
745.2
```



# Pandas DataFrames don't scale well

---

```
> df = pd.read_csv('flights-2016-01.xz',
                    dialect="excel")
```

DtypeWarning: Columns (77) have mixed types.  
Specify dtype option on import or set low\_memory=False.

CPU times: user 5.48 s, sys: 1.36 s, total: 6.83 s  
Wall time: 6.83 s

```
> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 445827 entries, 0 to 445826
Columns: 110 entries, Year to Unnamed: 109
dtypes: float64(71), int64(21), object(18)
memory usage: 374.2+ MB
```

```
> df.memory_usage(deep=True).sum() / 2**20
745.2
```

```
> df = pd.concat([
    pd.read_csv('flights-%s.xz' % m,
                dialect="excel")
    for m in ['2015-12','2016-01','2016-02']
])
```

DtypeWarning: Columns (48,76,77,84,85) have mixed types.  
DtypeWarning: Columns (77) have mixed types.  
DtypeWarning: Columns (77,84) have mixed types.  
Specify dtype option on import or set low\_memory=False.

CPU times: user 18.4 s, sys: 6.66 s, total: 25.1 s  
Wall time: 1min 26s

```
> df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1348946 entries, 0 to 423888
Columns: 110 entries, Year to Unnamed: 109
dtypes: float64(69), int64(21), object(20)
memory usage: 1.1+ GB
```

```
> df.memory_usage(deep=True).sum() / 2**20
2326.9
```



# Need to scale up in multiple places...

---

Data storage format

Load data in parallel

Parallelize intermediate calculations

Run on multiples core/machines

Effort required to write/optimize correct parallel code

And we still want to use Python and Pandas!



# Dask is a ‘drop-in’ replacement for pandas\*

---

```
> import dask.dataframe as dd  
  
> paths = [ 'flights-%s.xz' % dt.strftime('%Y-%m')  
    for dt in pd.date_range('2015-12', '2016-03', freq='M') ]  
  
> cols = ['FlightDate', 'Origin', 'Dest', 'OriginState', 'DestState',  
    'Carrier', 'FlightNum', 'TailNum', 'CRSDepTime', 'CRSArrTime',  
    'DepDelay', 'ArrDelay', 'Flights', 'Cancelled', 'Diverted', ]  
  
>ddf = dd.read_csv(  
    paths,  
    dialect="excel", encoding='latin-1',  
    header=0, usecols=cols,  
    compression='xz', blocksize=None,  
    parse_dates=['FlightDate'],  
    dtype={'FlightNum': str}, )
```



# Dask is a ‘drop-in’ replacement for pandas\*

---

```
> import dask.dataframe as dd

> paths = [ 'flights-%s.xz' % dt.strftime('%Y-%m')
    for dt in pd.date_range('2015-12', '2016-03', freq='M') ]

> cols = ['FlightDate', 'Origin', 'Dest', 'OriginState', 'DestState',
    'Carrier', 'FlightNum', 'TailNum', 'CRSDepTime', 'CRSArrTime',
    'DepDelay', 'ArrDelay', 'Flights', 'Cancelled', 'Diverted', ]

> ddf = dd.read_csv(
    paths,
    dialect="excel", encoding='latin-1',
    header=0, usecols=cols,
    compression='xz', blocksize=None,
    parse_dates=['FlightDate'],
    dtype={'FlightNum': str}, )

> ddf[['Carrier', 'Flights', 'Cancelled']].groupby('Carrier').sum()
CPU times: user 16 ms, sys: 0 ns, total: 16 ms
Wall time: 18.3 ms
```

457

---

\* ... for a subset of pandas DataFrame operations



# Dask is a ‘drop-in’ replacement for pandas\*

```
> import dask.dataframe as dd

> paths = [ 'flights-%s.xz' % dt.strftime('%Y-%m')
    for dt in pd.date_range('2015-12', '2016-03', freq='M') ]

> cols = ['FlightDate', 'Origin', 'Dest', 'OriginState', 'DestState',
    'Carrier', 'FlightNum', 'TailNum', 'CRSDepTime', 'CRSArrTime',
    'DepDelay', 'ArrDelay', 'Flights', 'Cancelled', 'Diverted', ]

>ddf = dd.read_csv(
    paths,
    dialect="excel", encoding='latin-1',
    header=0, usecols=cols,
    compression='xz', blocksize=None,
    parse_dates=['FlightDate'],
    dtype={'FlightNum': str}, )

> ddf[['Carrier','Flights','Cancelled']].groupby('Carrier').sum()
CPU times: user 16 ms, sys: 0 ns, total: 16 ms
Wall time: 18.3 ms

>_.compute()
CPU times: user 13.9 s, sys: 2.22 s, total: 16.1 s
Wall time: 9.62 s
```

	Flights	Cancelled
Carrier		
AA	223582.0	5001.0
AS	42063.0	311.0
B6	68137.0	1522.0
DL	208121.0	1472.0
EV	126036.0	4409.0
F9	21865.0	266.0
HA	18390.0	12.0
MQ	20993.0	805.0
NK	32072.0	894.0
OO	140825.0	3471.0
UA	122148.0	2470.0
VX	15859.0	241.0
WN	308855.0	5677.0

\* ... for a subset of pandas DataFrame operations



# Expressions build a dependency graph...

```
> task = ddf[['Carrier', 'Flights','Cancelled']]  
    .groupby('Carrier')  
    .sum()
```

```
> task.compute()  
CPU times: user 13.9 s,  
sys: 2.22 s, total: 16.1 s  
Wall time: 9.62 s
```

```
> task.visualize()
```



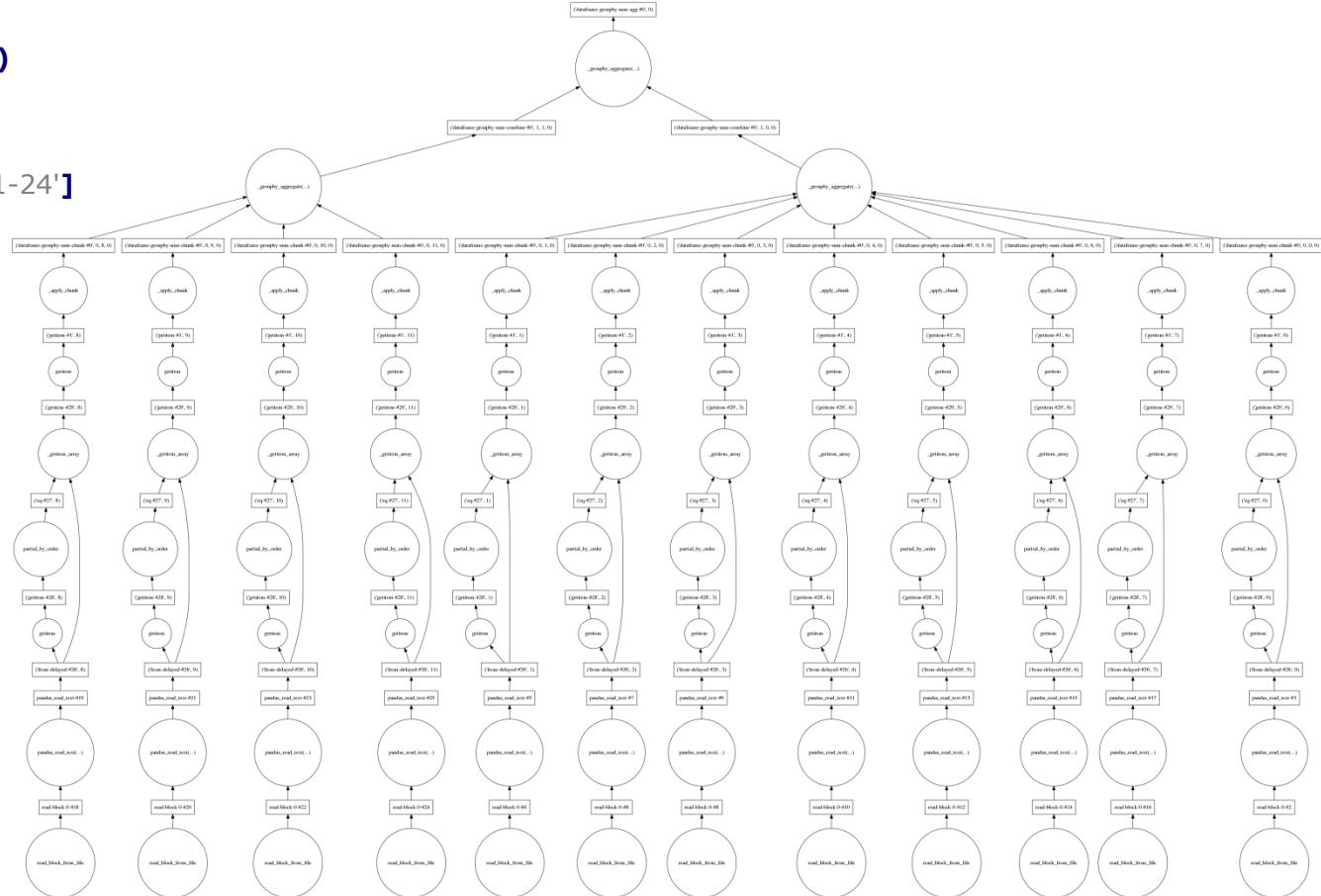


... that can get arbitrarily complex

```
> ddf = dd.read_csv('flights-* .xz', ...)
```

```
> task = ddf[ddf.FlightDate=='2016-01-24']
[[Carrier, 'Flights','Cancelled']]
.groupby('Carrier')
.sum()
```

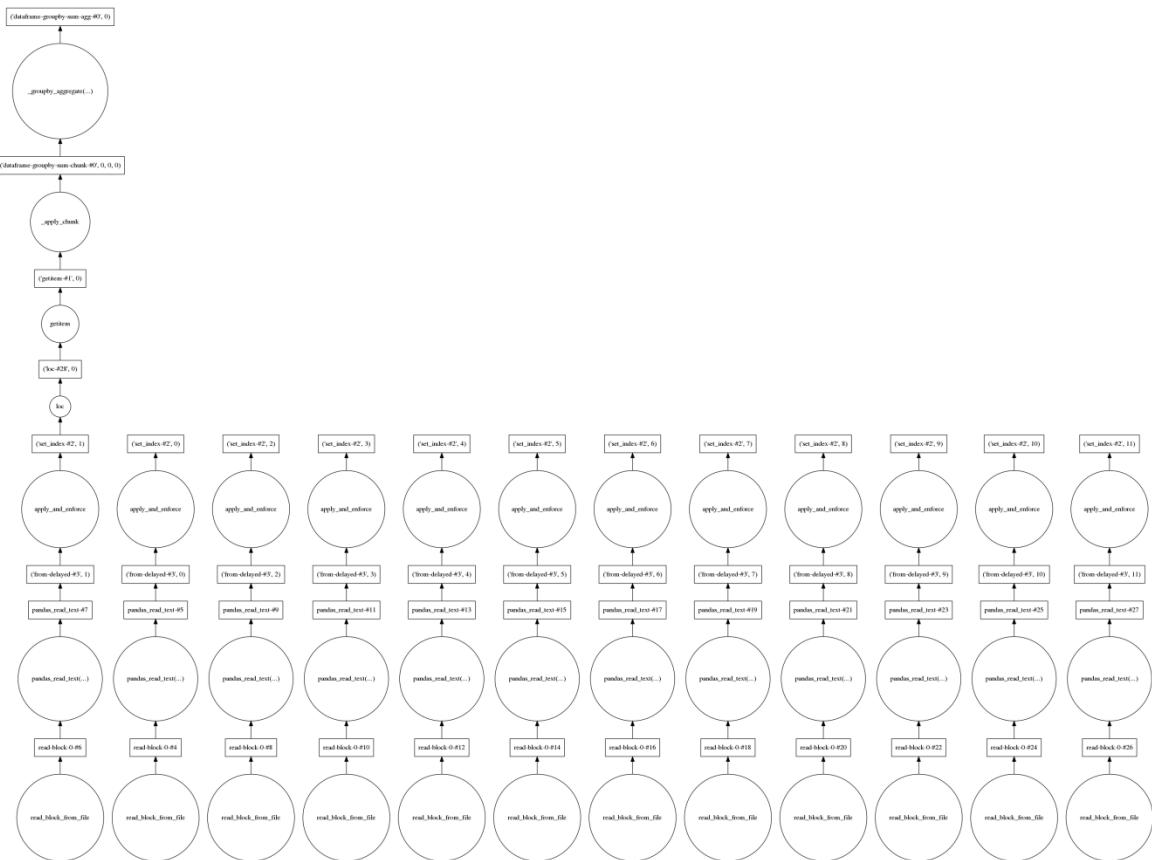
```
> task.visualize()
```





# Dask prunes the graph where it can

```
> ddf = dd.read_csv('flights-* .xz', ...)  
> ddf = ddf.set_index('FlightDate')  
  
> task = ddf['2016-01-24'][['Carrier', 'Flights', 'Cancelled']].groupby('Carrier').sum()  
  
> task.visualize()
```





# Storage formats like parquet are faster than csv

```
ddf = load_data(start='2016-01', end='2017-02')

def sort_partition(df):
    return df.set_index(df.FlightDate).sort_index()

task = ddf.map_partitions(func=sort_partition)

task.to_parquet('flights.parq',
                compression='SNAPPY')

$ ls -hs flights.parq/
total 100M
4.0K _common_metadata
20K _metadata
8.5M part.0.parquet
8.0M part.1.parquet
7.5M part.2.parquet
...
8.2M part.11.parquet
8.2M part.12.parquet
```



# Storage formats like parquet are faster than csv

```
ddf = load_data(start='2016-01', end='2017-02')

def sort_partition(df):
    return df.set_index(df.FlightDate).sort_index()

task = ddf.map_partitions(func=sort_partition)

task.to_parquet('flights.parq',
                compression='SNAPPY')

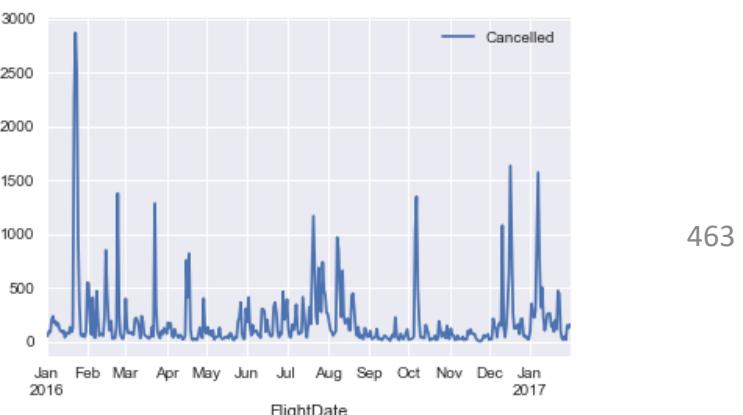
$ ls -hs flights.parq/
```

```
total 100M
4.0K _common_metadata
20K _metadata
8.5M part.0.parquet
8.0M part.1.parquet
7.5M part.2.parquet
...
8.2M part.11.parquet
8.2M part.12.parquet
```

## Parquet example 1 – whole dataset

```
ddf = dd.read_parquet('flights.parq',
                      columns=['Cancelled']) # 50 ms

task = ddf.groupby(ddf.index).sum()
out = task.compute() # 313 ms
out.plot()
```



463



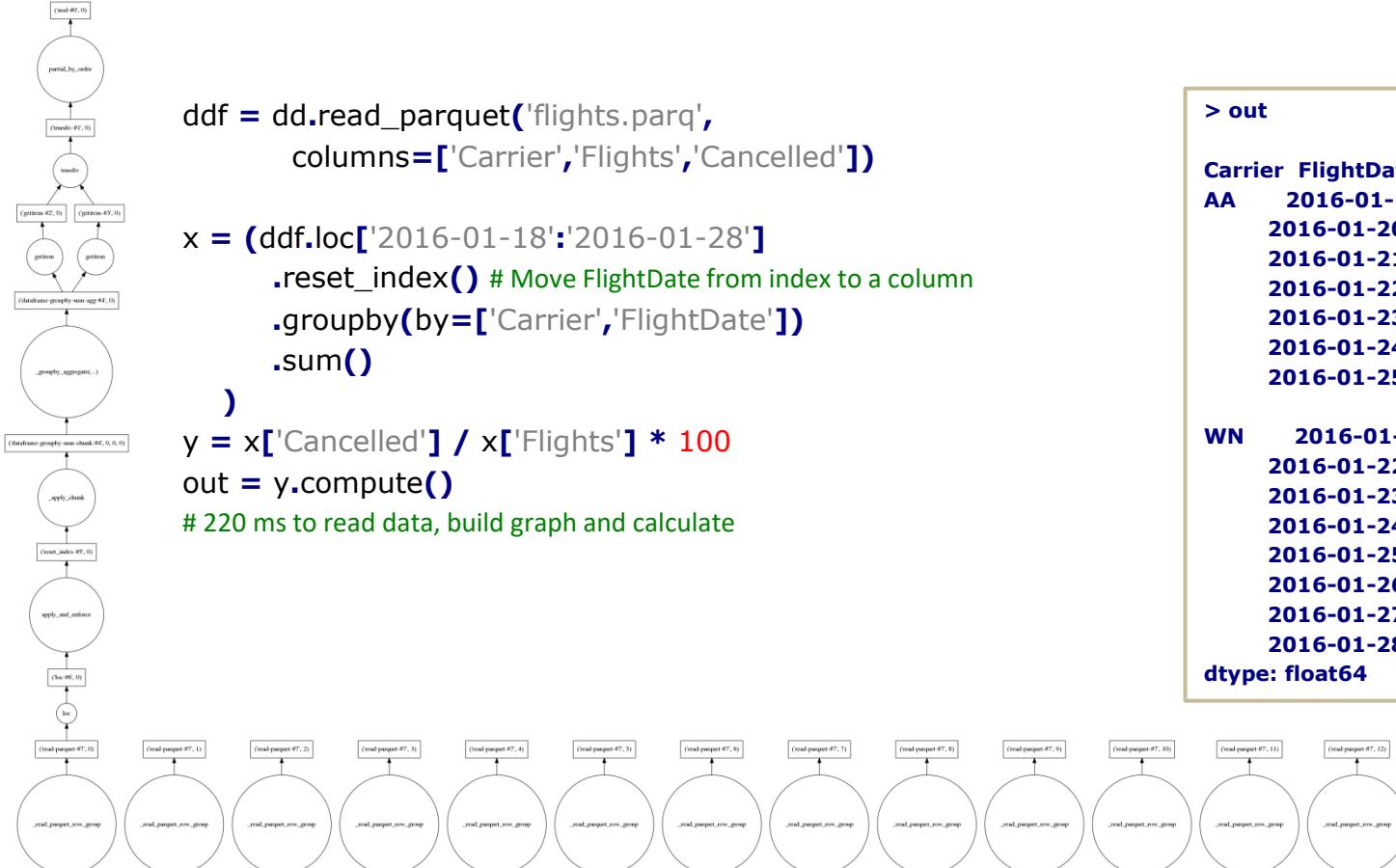
# Parquet example 2: read a subset of the partitions

```
ddf = dd.read_parquet('flights.parq',
                      columns=['Carrier','Flights','Cancelled'])

x = (ddf.loc['2016-01-18':'2016-01-28']
     .reset_index() # Move FlightDate from index to a column
     .groupby(by=['Carrier','FlightDate'])
     .sum()

y = x['Cancelled'] / x['Flights'] * 100
out = y.compute()

# 220 ms to read data, build graph and calculate
```



```
> out
```

Carrier	FlightDate	Cancelled
AA	2016-01-19	0.339992
	2016-01-20	0.160064
	2016-01-21	1.327606
	2016-01-22	30.712339
	2016-01-23	36.307838
	2016-01-24	25.588114
	2016-01-25	8.853119
	...	
WN	2016-01-21	1.133787
	2016-01-22	14.436219
	2016-01-23	21.369961
	2016-01-24	17.904074
	2016-01-25	4.630682
	2016-01-26	1.394422
	2016-01-27	0.910643
	2016-01-28	0.597610

**dtype: float64**



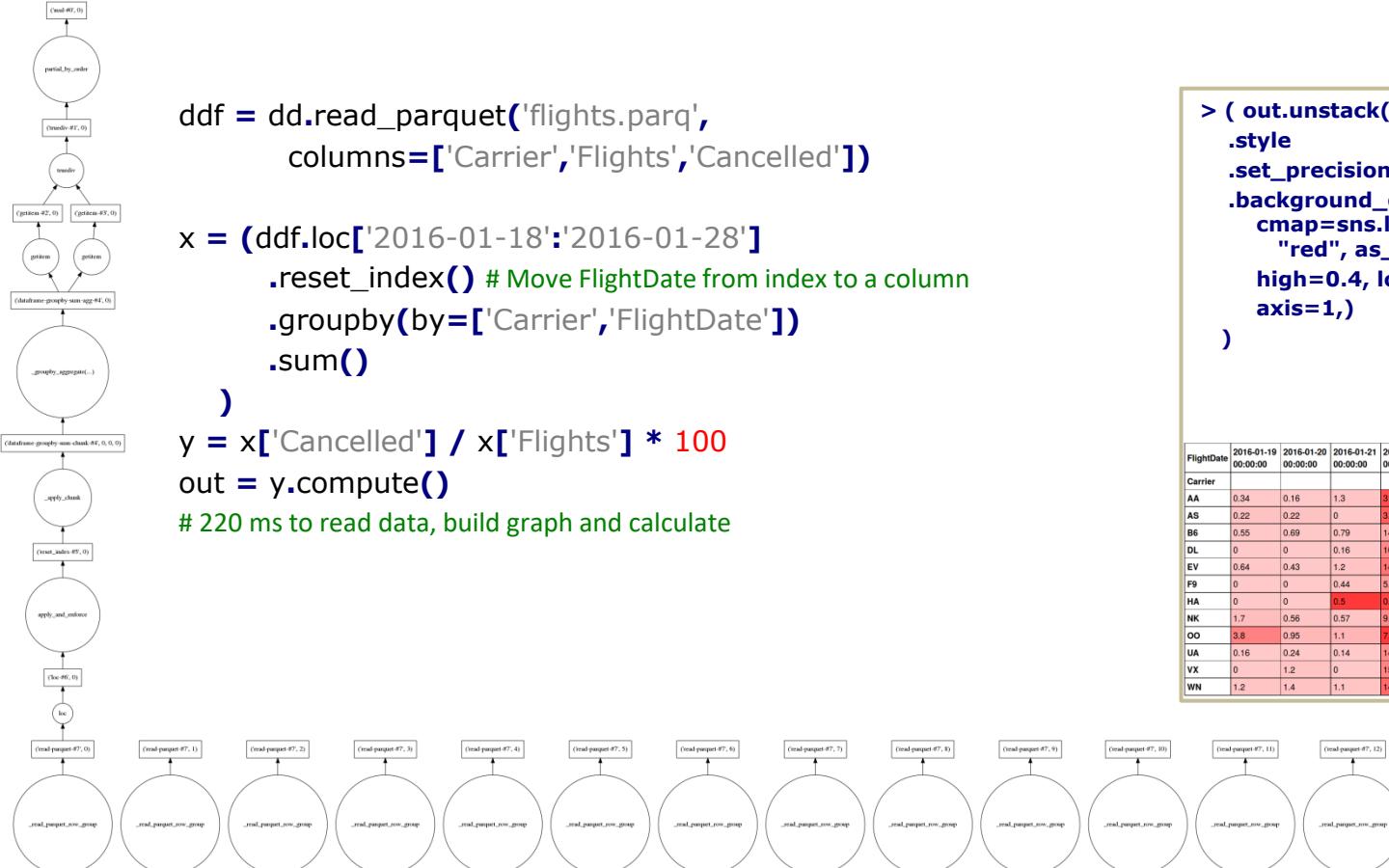
# Parquet example 2: read a subset of the partitions

```
ddf = dd.read_parquet('flights.parq',
                      columns=['Carrier','Flights','Cancelled'])

x = (ddf.loc['2016-01-18':'2016-01-28']
      .reset_index() # Move FlightDate from index to a column
      .groupby(by=['Carrier','FlightDate'])
      .sum()

y = x['Cancelled'] / x['Flights'] * 100
out = y.compute()

# 220 ms to read data, build graph and calculate
```



```
> (out.unstack('FlightDate')
  .style
  .set_precision(2)
  .background_gradient(
    cmap=sns.light_palette(
      "red", as_cmap=True),
    high=0.4, low=0.2,
    axis=1,)
```

FlightDate	2016-01-19 00:00:00	2016-01-20 00:00:00	2016-01-21 00:00:00	2016-01-22 00:00:00	2016-01-23 00:00:00	2016-01-24 00:00:00	2016-01-25 00:00:00	2016-01-26 00:00:00	2016-01-27 00:00:00
Carrier	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00
AA	0.34	0.16	1.3	31	36	26	8.9	0.68	0.36
AS	0.22	0.22	0	3.6	4.9	4.1	1.3	0.22	0.22
B6	0.55	0.69	0.79	14	63	33	2.5	0	0.14
DL	0	0	0.16	10	22	12	0.88	0	0
EV	0.64	0.43	1.2	14	23	20	15	12	0.5
F9	0	0	0.44	5.3	16	8	0.45	0	0
HA	0	0	0.5	0.47	0.5	0	0	0	0
NK	1.7	0.56	0.57	9.5	22	13	2.5	0	0.28
OO	3.8	0.95	1.1	7.6	3.4	2.5	1.3	0.38	0.38
UA	0.16	0.24	0.14	14	29	32	19	8.4	0.47
VX	0	1.2	0	15	30	27	6.1	0	0.58
WN	1.2	1.4	1.1	14	21	18	4.6	1.4	0.91

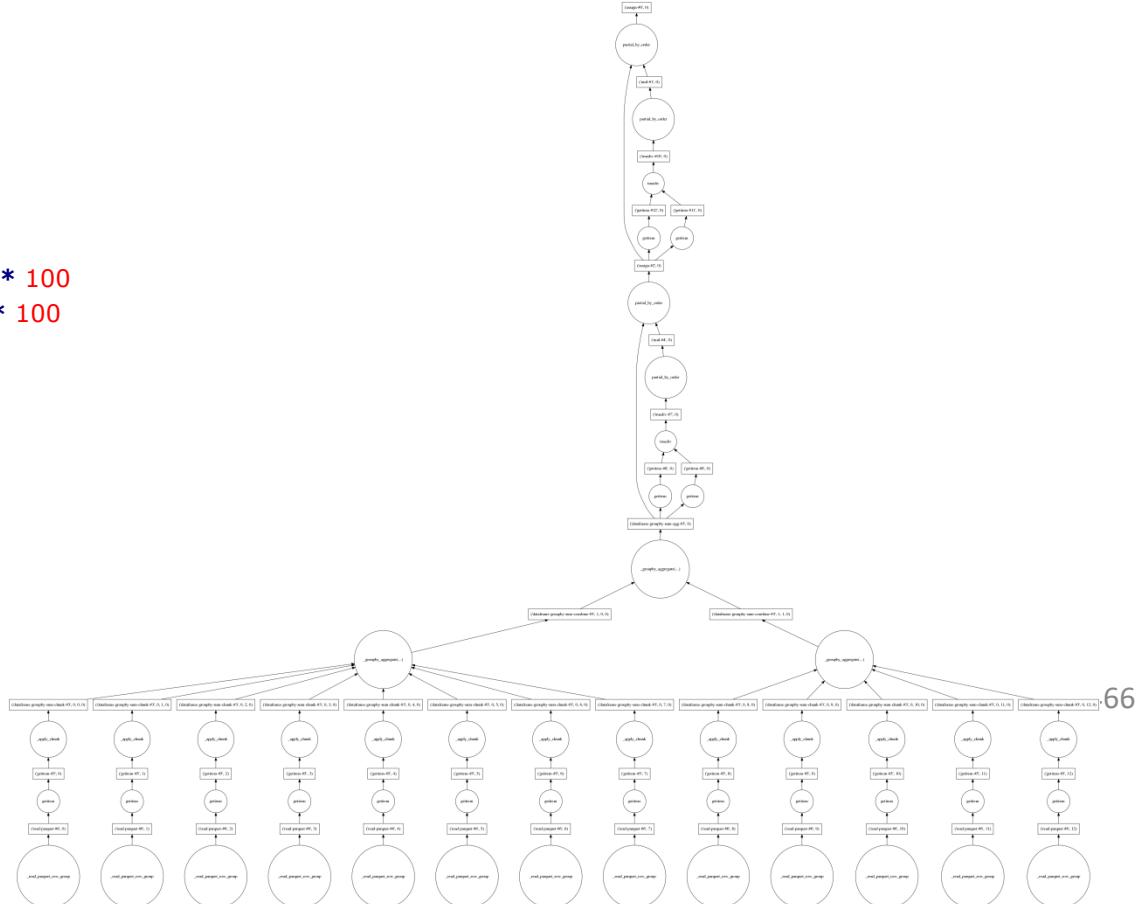


# Monitoring progress of a Dask calculation

```
ddf = dd.read_parquet('flights.parq')

sum_cols = ['Carrier', 'Flights', 'Cancelled', 'Diverted']
task = ddf[sum_cols].groupby('Carrier').sum()

task['CancelledPct'] = task['Cancelled'] / task['Flights'] * 100
task['DivertedPct'] = task['Diverted'] / task['Flights'] * 100
```





# Monitoring progress of a Dask calculation

```
ddf = dd.read_parquet('flights.parq')

sum_cols = ['Carrier', 'Flights', 'Cancelled', 'Diverted']
task = ddf[sum_cols].groupby('Carrier').sum()

task['CancelledPct'] = task['Cancelled'] / task['Flights'] * 100
task['DivertedPct'] = task['Diverted'] / task['Flights'] * 100
```

```
from dask.diagnostics import ProgressBar
```

```
with ProgressBar():
    out = task.compute()
```

```
[ ] | 0% Completed | 0.0s
```

```
>
```



# Monitoring progress of a Dask calculation

```
ddf = dd.read_parquet('flights.parq')

sum_cols = ['Carrier', 'Flights', 'Cancelled', 'Diverted']
task = ddf[sum_cols].groupby('Carrier').sum()

task['CancelledPct'] = task['Cancelled'] / task['Flights'] * 100
task['DivertedPct'] = task['Diverted'] / task['Flights'] * 100
```

```
from dask.diagnostics import ProgressBar

with ProgressBar():
    out = task.compute()
```

```
[#####] | 87% Completed | 2.4s
```

```
>
```



# Monitoring progress of a Dask calculation

```
ddf = dd.read_parquet('flights.parq')

sum_cols = ['Carrier', 'Flights', 'Cancelled', 'Diverted']
task = ddf[sum_cols].groupby('Carrier').sum()

task['CancelledPct'] = task['Cancelled'] / task['Flights'] * 100
task['DivertedPct'] = task['Diverted'] / task['Flights'] * 100

from dask.diagnostics import ProgressBar

with ProgressBar():
    out = task.compute()
```

```
> print(out)
```

Carrier	Flights	Cancelled	Diverted	CancelledPct	DivertedPct
AA	987627.0	11847.0	2421.0	1.199542	0.245133
AS	191991.0	1072.0	520.0	0.558360	0.270846
B6	307075.0	4322.0	774.0	1.407474	0.252056
DL	992559.0	4898.0	1923.0	0.493472	0.193742
EV	526027.0	13048.0	1723.0	2.480481	0.327550
F9	102881.0	1341.0	169.0	1.303448	0.164267
HA	83065.0	136.0	91.0	0.163727	0.109553
NK	150769.0	3070.0	218.0	2.036228	0.144592
OO	656079.0	10326.0	2181.0	1.573896	0.332429
UA	587470.0	5702.0	1522.0	0.970603	0.259077
VX	74903.0	806.0	272.0	1.076058	0.363136
WN	1407229.0	18179.0	3324.0	1.291830	0.236209

```
[#####] |100% Completed | 2.6s
```

469



# Profiling a Dask calculation

---

```
ddf = dd.read_parquet('flights.parq')

sum_cols = ['Carrier', 'Flights', 'Cancelled', 'Diverted']
task = ddf[sum_cols].groupby('Carrier').sum()

task['CancelledPct'] = task['Cancelled'] / task['Flights'] * 100
task['DivertedPct'] = task['Diverted'] / task['Flights'] * 100

from dask.diagnostics import Profiler, ResourceProfiler, CacheProfiler
from cachey import nbytes

with (Profiler() as prof, ResourceProfiler(dt=0.25) as rprof,
      CacheProfiler(metric=nbytes) as cprof):
    df = task.compute()
```



# Profiling a Dask calculation

```
ddf = dd.read_parquet('flights.parq')

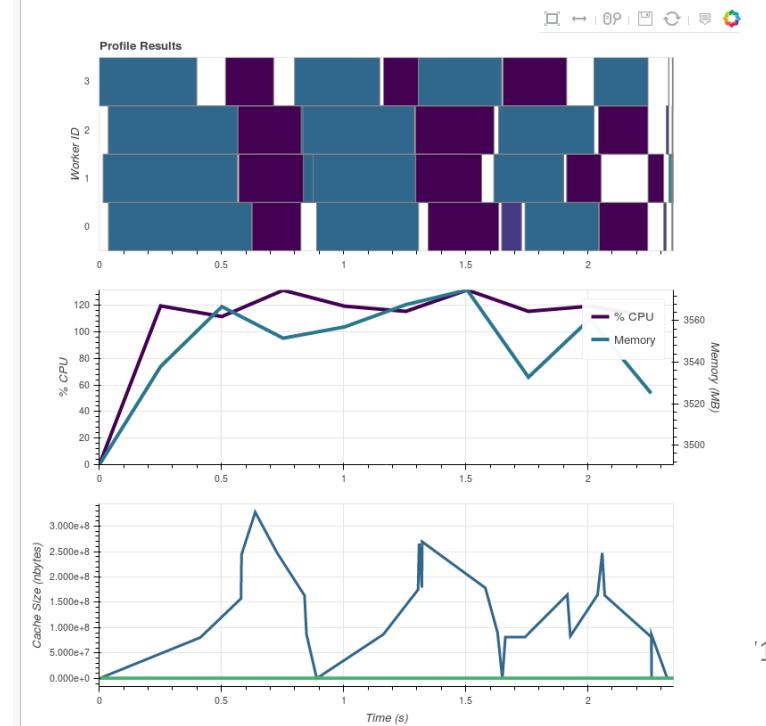
sum_cols = ['Carrier', 'Flights', 'Cancelled', 'Diverted']
task = ddf[sum_cols].groupby('Carrier').sum()

task['CancelledPct'] = task['Cancelled'] / task['Flights'] * 100
task['DivertedPct'] = task['Diverted'] / task['Flights'] * 100

from dask.diagnostics import Profiler, ResourceProfiler, CacheProfiler
from cachey import nbytes

with (Profiler() as prof, ResourceProfiler(dt=0.25) as rprof,
      CacheProfiler(metric=nbytes) as cprof):
    df = task.compute()

dask.diagnostics.visualize([prof, rprof, cprof], save=False, show=True)
```





# So what exactly is a Dask DataFrame?

```
> print(dd.DataFrame.__doc__)
```

Implements out-of-core DataFrame as a sequence of pandas DataFrames

## Parameters

-----

**dask**: dict

The dask graph to compute this DataFrame

**name**: str

The key prefix that specifies which keys in the dask comprise this particular DataFrame

**meta**: pandas.DataFrame

An empty ``pandas.DataFrame`` with names, dtypes, and index matching the expected output.

**divisions**: tuple of index values

Values along which we partition our blocks on the index

```
def from_pandas(data, npartitions=None, chunksize=None, sort=True, name=None):
```

"""

Construct a Dask DataFrame from a Pandas DataFrame

This splits an in-memory Pandas dataframe into several parts and constructs a dask.dataframe from those parts on which Dask.dataframe can operate in parallel.

Note that, despite parallelism, Dask.dataframe may not always be faster than Pandas. We recommend that you stay with Pandas for as long as possible before switching to Dask.dataframe.

## Parameters

-----

**data** : pandas.DataFrame or pandas.Series

The DataFrame/Series with which to construct a Dask DataFrame/Series

**npartitions** : int, optional

The number of partitions of the index to create. Note that depending on the size and index of the dataframe, the output may have fewer partitions than requested.

**chunksize** : int, optional

The size of the partitions of the index.

**sort**: bool

Sort input first to obtain cleanly divided partitions or don't sort and don't get cleanly divided partitions

**name**: string, optional

An optional keyname for the dataframe. Defaults to hashing the input

## Returns

-----

dask.DataFrame or dask.Series

A dask DataFrame/Series partitioned along the index



# The simplest Dask DataFrame

```
>>> df = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9],  
    [10,11,12],[13,14,15]], columns=['a','b','c'])
```

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12
4	13	14	15

```
>>> ddf = dd.from_pandas(df, npartitions=1)
```

	a	b	c
npartitions=1	int64	int64	int64
0	...	...	...
4	...	...	...

Dask Name: from\_pandas, 2 tasks

```
>>> ddf.divisions  
(0, 4)
```

```
>>> ddf._meta  
Empty DataFrame  
Columns: [a, b, c]  
Index: []
```

```
> print(dd.DataFrame.__doc__)  
dask: dict  
    The dask graph to compute this DataFrame  
name: str  
    The key prefix that specifies which keys in the dask  
    comprise this particular DataFrame  
meta: pandas.DataFrame  
    An empty ``pandas.DataFrame`` with names, dtypes, and  
    index matching the expected output.  
divisions: tuple of index values  
    Values along which we partition our blocks on the index
```

```
>>> ddf._name  
'from_pandas-b71f6a90'
```

```
>>> ddf.dask  
{  
    ('from_pandas-b71f6a90', 0): df,
```

```
}
```

```
>>> ddf.visualize()
```

```
('from_pandas-#0', 0)
```



# The next simplest Dask DataFrame

```
>>> df = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9],  
    [10,11,12],[13,14,15]], columns=['a','b','c'])
```

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12
4	13	14	15

```
>>> ddf = dd.from_pandas(df, npartitions=2)
```

	a	b	c
npartitions=1	int64	int64	int64
0	...	...	...
3	...	...	...
4	...	...	...

Dask Name: from\_pandas, 2 tasks

```
>>> ddf.divisions  
(0, 3, 4)
```

```
>>> ddf._meta  
Empty DataFrame  
Columns: [a, b, c]  
Index: []
```

```
> print(dd.DataFrame.__doc__)  
dask: dict  
    The dask graph to compute this DataFrame  
name: str  
    The key prefix that specifies which keys in the dask  
    comprise this particular DataFrame  
meta: pandas.DataFrame  
    An empty ``pandas.DataFrame`` with names, dtypes, and  
    index matching the expected output.  
divisions: tuple of index values  
    Values along which we partition our blocks on the index
```

```
>>> ddf._name  
'from_pandas-de36e0f9'
```

```
>>> ddf.dask  
{  
    ('from_pandas-de36e0f9', 0): df[0:3],  
    ('from_pandas-de36e0f9', 1): df[3:],  
}
```

```
>>> ddf.visualize()
```

('from\_pandas-#0', 0)

('from\_pandas-#0', 1)



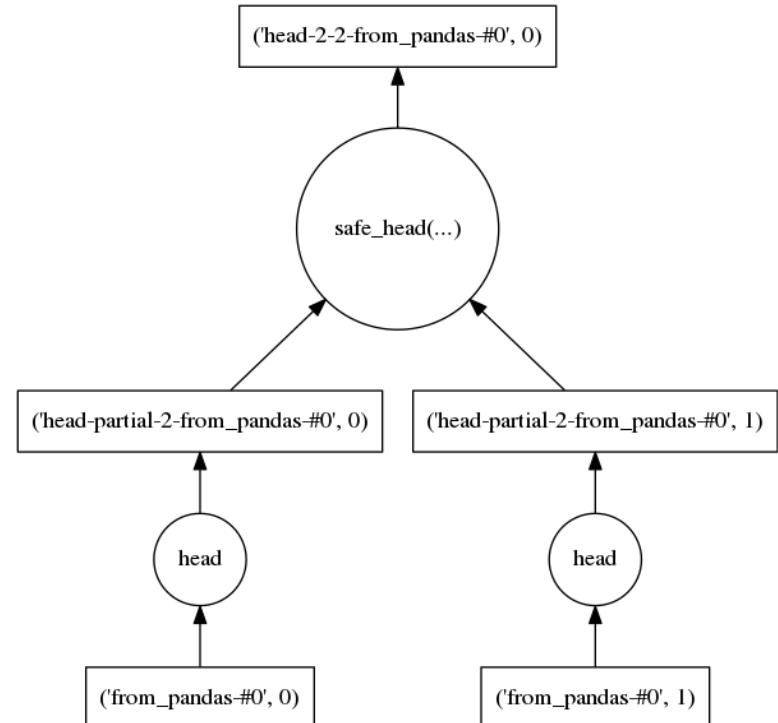
# dd.DataFrame.dask

```
>>> ddf = dd.from_pandas(df, npartitions=2)
       .head(n=2, npartitions=2, compute=False)

>>> ddf._name
'head-2-2-from_pandas-de36e0f9b'

>>> ddf.dask
{
('head-2-2-from_pandas-de36e0f9b', 0):
    (dask.dataframe.core.safe_head,
     (function dask.dataframe.core._concat,
      [('head-partial-2-from_pandas-de36e0f9b', 0),
       ('head-partial-2-from_pandas-de36e0f9b', 1)])
     ), 2
    ),
('head-partial-2-from_pandas-de36e0f9b', 0):
    (<methodcaller: head>, ('from_pandas-de36e0f9b', 0), 2),
('head-partial-2-from_pandas-de36e0f9b', 1):
    (<methodcaller: head>, ('from_pandas-de36e0f9b', 1), 2),
('from_pandas-de36e0f9b', 0): df[0:3],
('from_pandas-de36e0f9b', 1): df[3:],
}

>>> ddf.keys()
[('head-2-2-from_pandas-de36e0f9b', 0)]
>>> ddf.compute()
```





# Reimplement pandas methods lazily

```
>>> ddf = dd.from_pandas(df, npartitions=2)
       .head(n=2, npartitions=2, compute=False)

>>> ddf._name
'head-2-2-from_pandas-de36e0f9b'

>>> ddf.dask
{
('head-2-2-from_pandas-de36e0f9b', 0):
    (dask.dataframe.core.safe_head,
     (function dask.dataframe.core._concat,
      [('head-partial-2-from_pandas-de36e0f9b', 0),
       ('head-partial-2-from_pandas-de36e0f9b', 1)]
      ), 2
    ),
('head-partial-2-from_pandas-de36e0f9b', 0):
    (<methodcaller: head>, ('from_pandas-de36e0f9b', 0), 2),
('head-partial-2-from_pandas-de36e0f9b', 1):
    (<methodcaller: head>, ('from_pandas-de36e0f9b', 1), 2),
('from_pandas-de36e0f9b', 0): df[0:3],
('from_pandas-de36e0f9b', 1): df[3:],
}

>>> ddf._keys()
[('head-2-2-from_pandas-de36e0f9b', 0)]
>>> ddf.compute()
```

dask.dataframe.core.\_Frame (L758+):

```
def head(self, n=5, npartitions=1, compute=True):
    """ First n rows of the dataset"""
    if npartitions <= -1:
        npartitions = self.npartitions

    name = 'head-%d-%d-%s' % (npartitions, n, self._name)

    if npartitions > 1:
        name_p = 'head-partial-%d-%s' % (n, self._name)
        dsk = {}
        for i in range(npartitions):
            dsk[(name_p, i)] = (M.head, (self._name, i), n)
        concat = (_concat, [(name_p, i) for i in range(npartitions)])
        dsk[(name, 0)] = (safe_head, concat, n)
    else:
        dsk = {(name, 0): (safe_head, (self._name, 0), n)}

    result = new_dd_object(merge(self.dask, dsk), name, self._meta,
                          [self.divisions[0], self.divisions[npartitions]])

    if compute:
        result = result.compute()
    return result
```

476



# Now we execute the graph with 'compute'

1. Optimize
  - Cull – remove unnecessary tasks
  - Fuse tasks – make parallelization less granular
  - Inline cheap functions
2. Get graphs keys to evaluate
3. Execute in parallel with scheduler
  - ‘get’ function
  - Sort nodes
  - Balance work between threads, processes, cores, over a cluster
4. Optionally cache intermediate results

References in dask source code:

- optimize.py, order.py and async.py

```
dask.dataframe.base (L139+):
```

```
def compute(*args, **kwargs):  
    """Compute several dask collections at once.  
    args : Any dask objects are computed and the result is returned.  
    traverse : Set to False to not look for dask objects in Python collections.  
    get : An optional alternative scheduler ``get`` function to use.  
    optimize_graph : If True [default], optimize the graph before computation.  
                    Otherwise run as is. This can be useful for debugging.  
    kwargs : Extra keywords to forward to the scheduler ``get`` function.  
    """  
  
    from dask.delayed import delayed  
    traverse = kwargs.pop('traverse', True)  
  
    if traverse:  
        args = tuple(delayed(a) if isinstance(a,  
                                              (list, set, tuple, dict, Iterator))  
                     else a for a in args)  
  
    optimize_graph = kwargs.pop('optimize_graph', True)  
    variables = [a for a in args if isinstance(a, Base)]  
    if not variables:  
        return args  
  
    get = kwargs.pop('get', None) or __globals__['get']  
    dsk = collections_to_dsk(variables, optimize_graph, **kwargs)  
    keys = [var._keys() for var in variables]  
    results = get(dsk, keys, **kwargs)  
  
    results_iter = iter(results)  
    return tuple(a if not isinstance(a, Base)  
                else a._finalize(next(results_iter))  
                for a in args)
```



---

# Scikit-Learn

## Tema 10



# Python scikit-learn

---

Popular machine learning toolkit in Python <http://scikit-learn.org/stable/>  
Requirements

Anaconda

Available from <https://www.continuum.io/downloads>

Includes numpy, scipy, and scikit-learn (former two are necessary for scikit-learn)

---

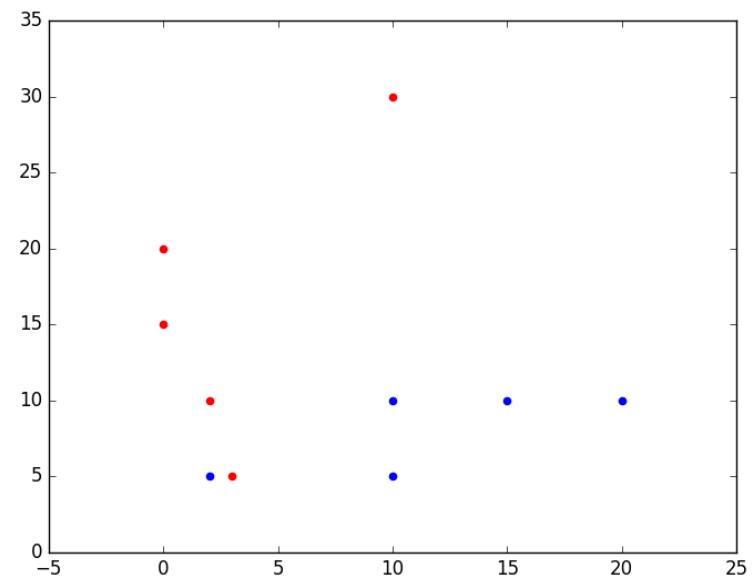


# Data

We think of data as vectors in a fixed dimensional space

For example

Heart disease	Cigarettes per day	Exercise per day (mins)
1	10	10
1	2	5
1	20	10
1	10	5
1	15	10
0	10	30
0	2	10
0	3	5
0	0	20
0	0	15





# Classification

---

Widely used task: given data determine the class it belongs to. The class will lead to a decision or outcome.

Used in many different places:

DNA and protein sequence classification

Insurance

Weather

Experimental physics: Higgs Boson determination



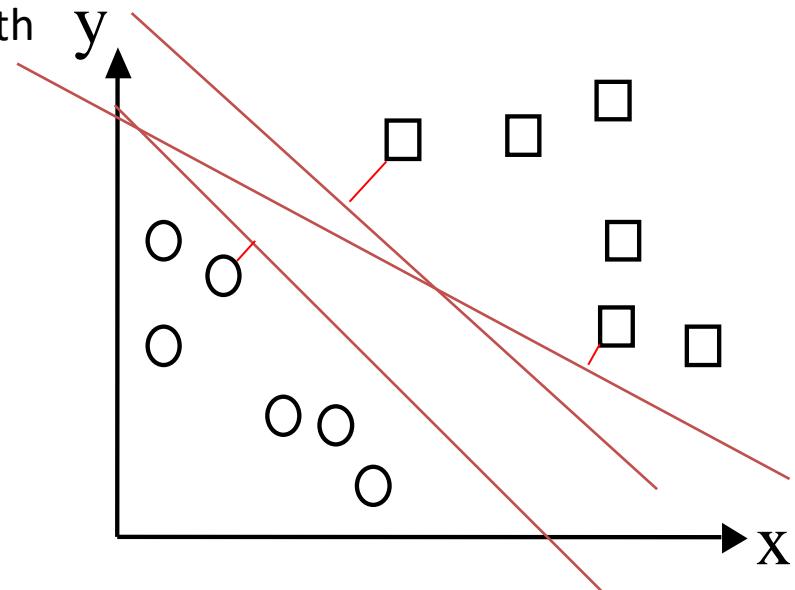
# Linear models

---

We think of a linear model as a hyperplane in space

The margin is the minimum distance of all closest point (misclassified have negative distance)

The support vector machine is the hyperplane with largest margin





# Support vector machine: optimally separating hyperplane

---

In practice we allow for error terms in case there is no hyperplane.

$$\min_{w, w_0, \chi_i} \left( \frac{1}{2} \|w\|^2 + C \sum_i \chi_i \right)$$

s.t.

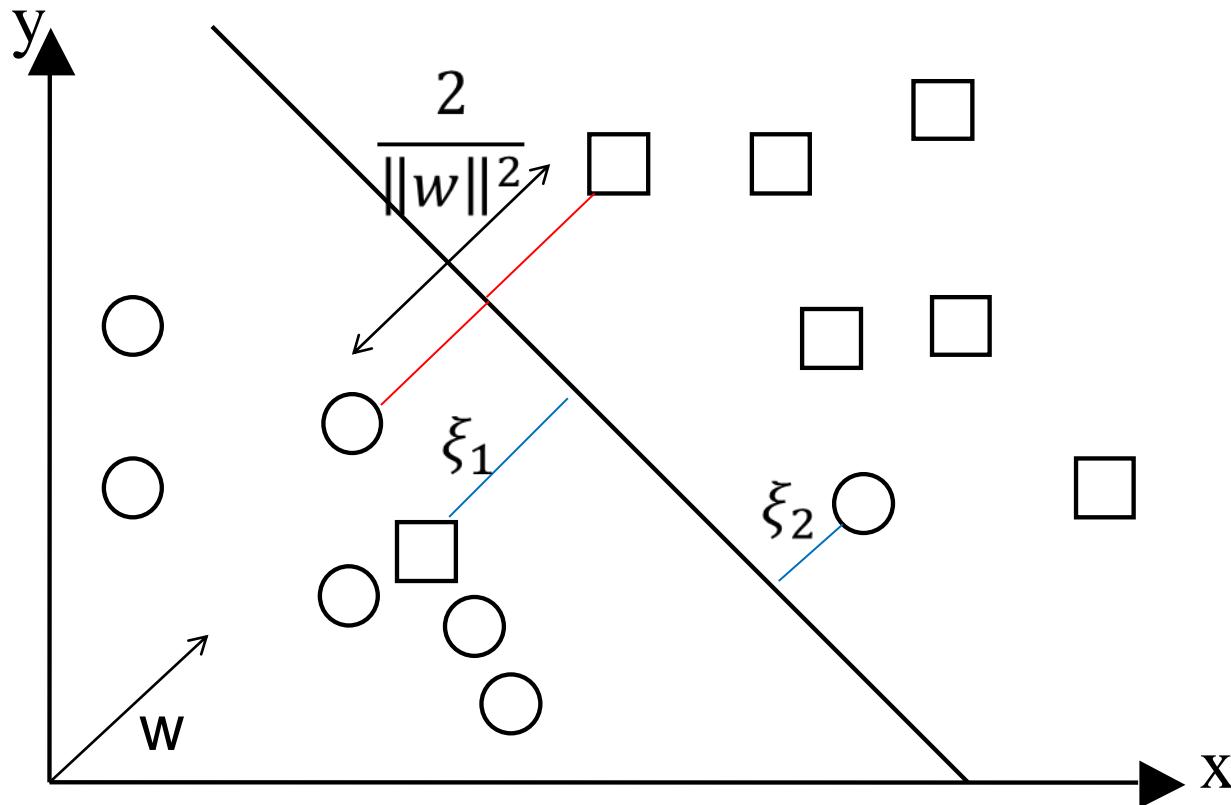
$$y_i (w^T x + w_0) \geq 1 - \chi_i$$

$$\chi_i \geq 0$$



# Optimally separating hyperplane with errors

---



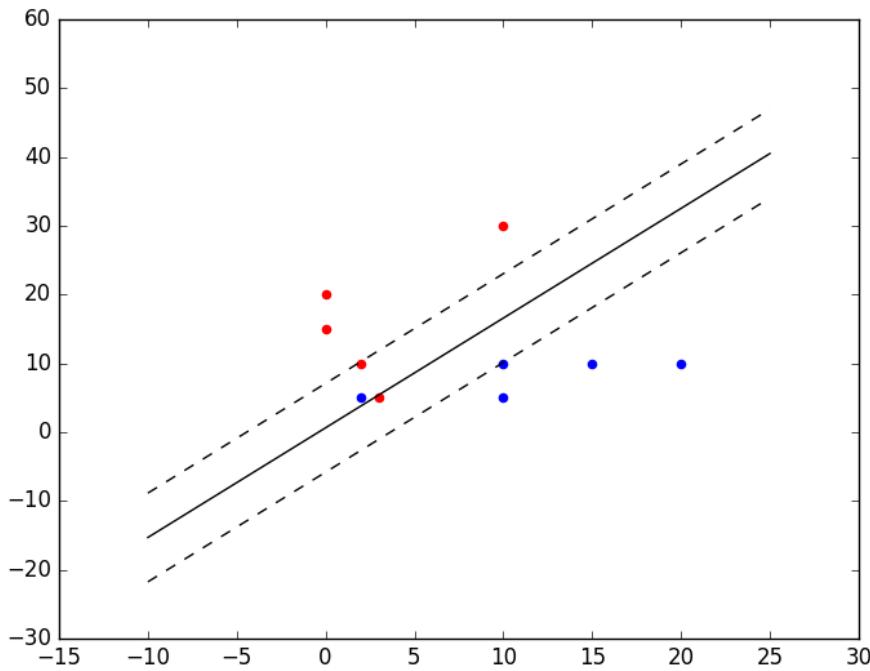


# SVM on simple data

---

Run SVM on example data shown earlier

Solid line is SVM and dashed indicates margin





# SVM in scikit-learn

---

Dataset are taken from the UCI machine learning repository

Learn an SVM model on training data

Which parameter settings?

C: tradeoff between error and model complexity (margin)

max\_iter: depth of the gradient descent algorithm

Predict on test data



# SVM in scikit-learn

---

## Analysis of SVM program on breast cancer data

```
import numpy as np
from sklearn import svm

f = open("bc.train.0")
data = np.loadtxt(f)
train = data[:,1:]
trainlabels = data[:,0]

f = open("bc.test.0")
data = np.loadtxt(f)
test = data[:,1:]
testlabels = data[:,0]

clf = svm.LinearSVC()
clf.fit(train,trainlabels)
prediction = clf.predict(test)
print(prediction)

err = 0
for i in range(0, len(prediction), 1):
    if(prediction[i] != testlabels[i]):
        err += 1
err = err/len(testlabels)
```

---



# Non-linear classification

---

In practice some datasets may not be classifiable.  
Remember this may not be a big deal because the test  
error is more important than the train one



# Non-linear classification

---

## Neural networks

Create a new representation of the data where it is linearly separable

Large networks leads to deep learning

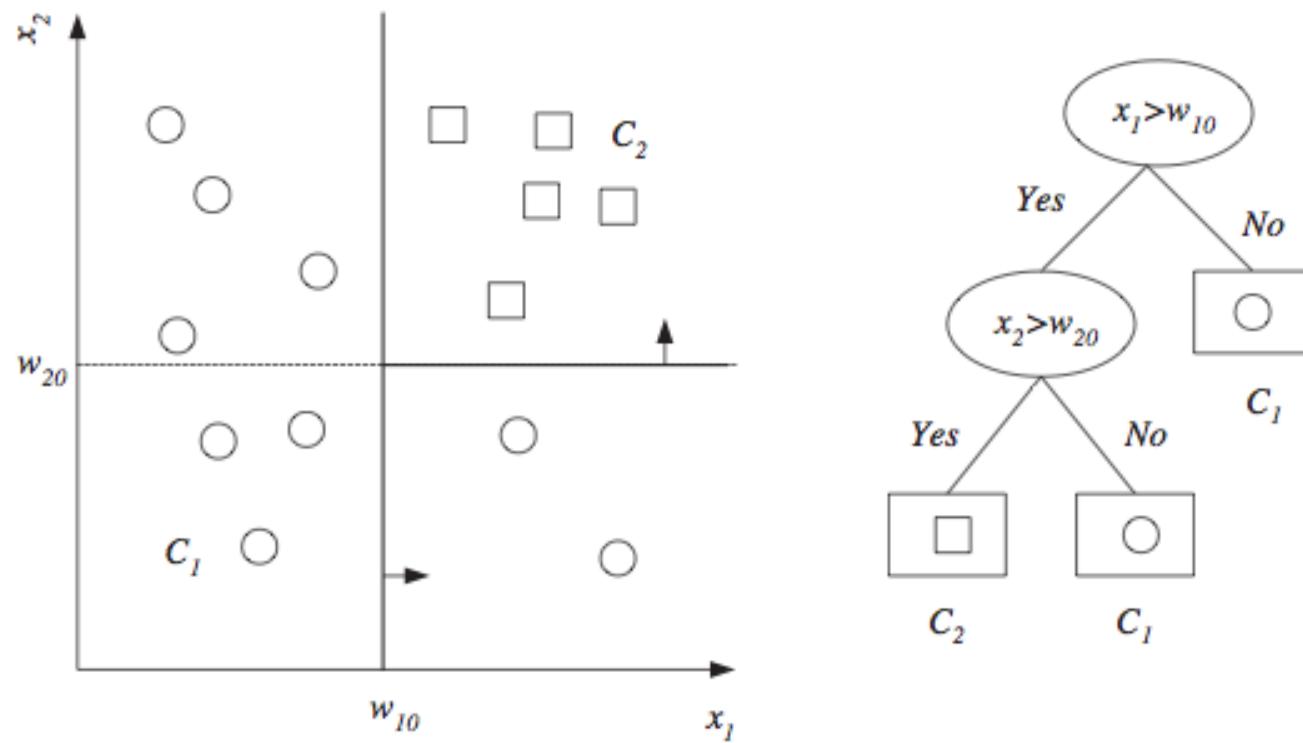
## Decision trees

Use several linear hyperplanes arranged in a tree

Ensembles of decision trees are state of the art such as random forest and boosting

---

# Decision tree



**Figure 9.1** Example of a dataset and the corresponding decision tree. Oval nodes are the decision nodes and rectangles are leaf nodes. The univariate decision node splits along one axis, and successive splits are orthogonal to each other. After the first split,  $\{x | x_1 < w_{10}\}$  is pure and is not split further.



# Combining classifiers by bagging

---

A single decision tree can overfit the data and have poor generalization (high error on test data). We can relieve this by bagging

## Bagging

Randomly sample training data by bootstrapping

Determine classifier  $C_i$  on sampled data

Goto step 1 and repeat  $m$  times

For final classifier output the majority vote

## Similar to tree bagging

Compute decision trees on bootstrapped datasets

Return majority vote



# Variance reduction by voting

---

What is the variance of the output of k classifiers?

Thus we want classifiers to be independent to minimize variance

Given

increas  
of Pat

$$\text{Var}(y) = \frac{1}{L^2} \text{Var} \left( \sum_j d_j \right) = \frac{1}{L^2} \left[ \sum_j \text{Var}(d_j) + 2 \sum_j \sum_{i < j} \text{Cov}(d_j, d_i) \right]$$

is



# Random forest

---

In addition to sampling datapoints (feature vectors) we also sample features (to increase independence among classifiers)

Compute many decision trees and output majority vote

Can also rank features

Alternative to bagging is to select datapoints with different probabilities that change in the algorithm (called boosting)

---



# Decision tree and random forest in scikit-learn

---

Learn a decision tree and random forest on training data

Which parameter settings?

Decision tree:

- Depth of tree

Random forest:

- Number of trees

- Percentage of columns

Predict on test data



# Decision tree and random forest in scikit-learn

---

```
From sklearn import tree
from sklearn.ensemble import RandomForestClassifier
import numpy as np

f = open("bc.train.0")
data = np.loadtxt(f)
train = data[:,1:]
trainlabels = data[:,0]

f = open("bc.test.0")
data = np.loadtxt(f)
test = data[:,1:]
testlabels = data[:,0]

clf = tree.DecisionTreeClassifier()
clf.fit(train,trainlabels)
prediction = clf.predict(test)

rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(train,trainlabels)
prediction2 = rfc.predict(test)

err = 0
err2 = 0
for i in range(0, len(prediction), 1):
    if(prediction[i] != testlabels[i]):
        err += 1
    if(prediction2[i] != testlabels[i]):
        err2 += 1
err = err/len(testlabels)
err2 = err2/len(testlabels)
print(err,err2)
```

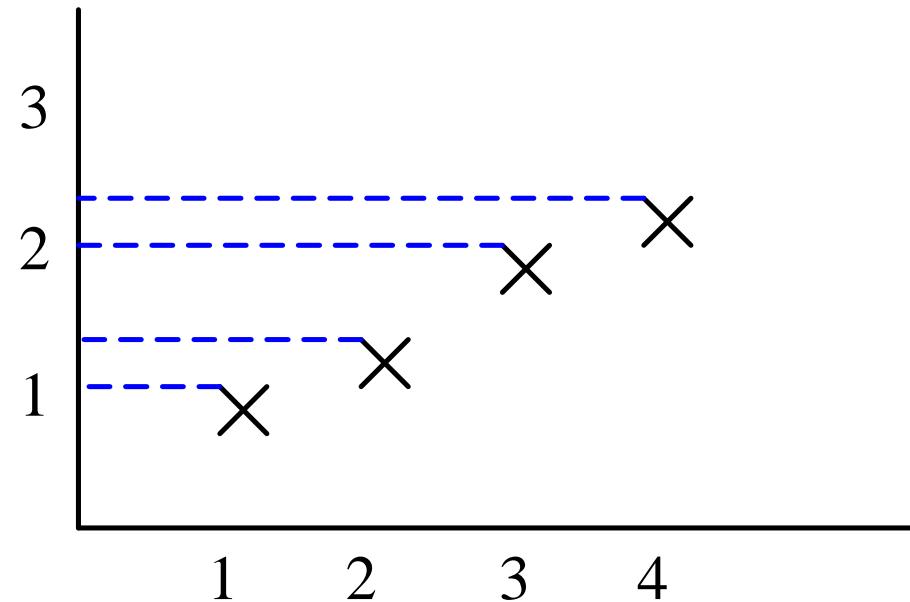
---



# Data projection

---

What is the mean and variance here?

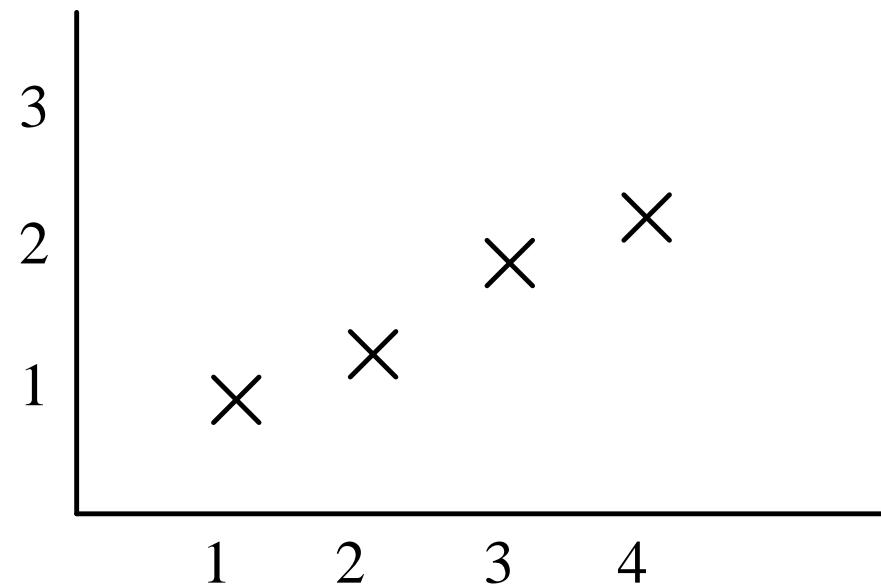




# Data projection

---

Which line maximizes variance?

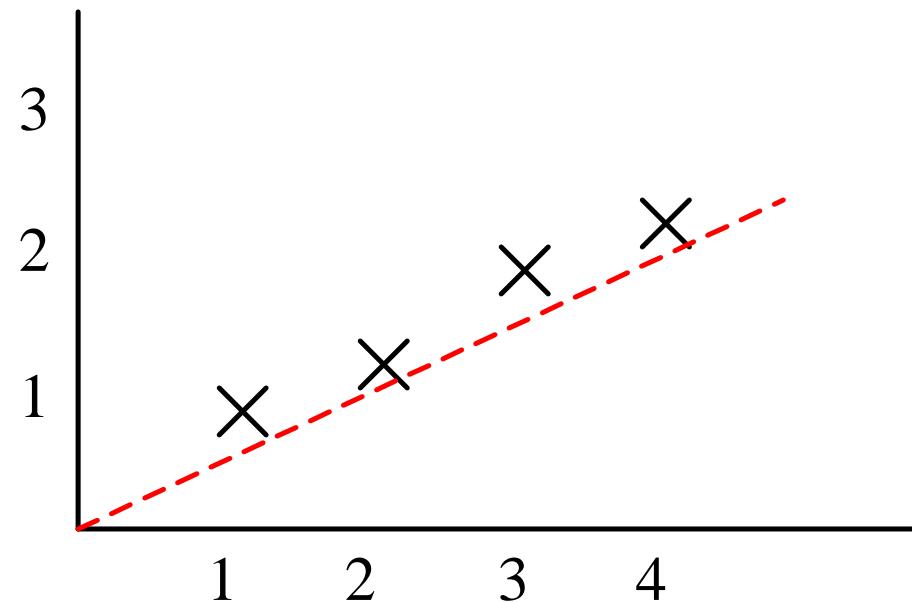




# Data projection

---

Which line maximizes variance?





# Principal component analysis

---

Find vector  $w$  of length 1 that maximizes variance of projected data

---



# PCA optimization problem

---

$$\arg \max_w \frac{1}{n} \sum_{i=1}^n (w^T x_i - w^T m)^2 \text{ subject to } w^T w = 1$$

The optimization criterion can be rewritten as

$$\arg \max_w \frac{1}{n} \sum_{i=1}^n (w^T (x_i - m))^2 =$$

$$\arg \max_w \frac{1}{n} \sum_{i=1}^n (w^T (x_i - m))^T (w^T (x_i - m)) =$$

$$\arg \max_w \frac{1}{n} \sum_{i=1}^n ((x_i - m)^T w)(w^T (x_i - m)) =$$

$$\arg \max_w \frac{1}{n} \sum_{i=1}^n w^T (x_i - m)(x_i - m)^T w =$$

$$\arg \max_w w^T \frac{1}{n} \sum_{i=1}^n (x_i - m)(x_i - m)^T w =$$

---

$$\arg \max_w w^T \sum_w w \text{ subject to } w^T w = 1$$

# Dimensionality reduction and visualization with PCA

```
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA

import numpy as np

f = open("bc")
data = np.loadtxt(f)

X = data[:,1:]
Y = data[:,0]

pca = PCA(n_components=2)
pca.fit(X)
newdata = pca.transform(X)

y1 = []
y2 = []
for i in range(0, len(newdata), 1):
    if(Y[i] == 1):
        y1.append(newdata[i][0])
        y2.append(newdata[i][1])

plt.scatter(y1, y2, color='blue')

y1 = []
y2 = []
for i in range(0, len(newdata), 1):
    if(Y[i] == -1):
        y1.append(newdata[i][0])
        y2.append(newdata[i][1])

plt.scatter(y1, y2, color='red')

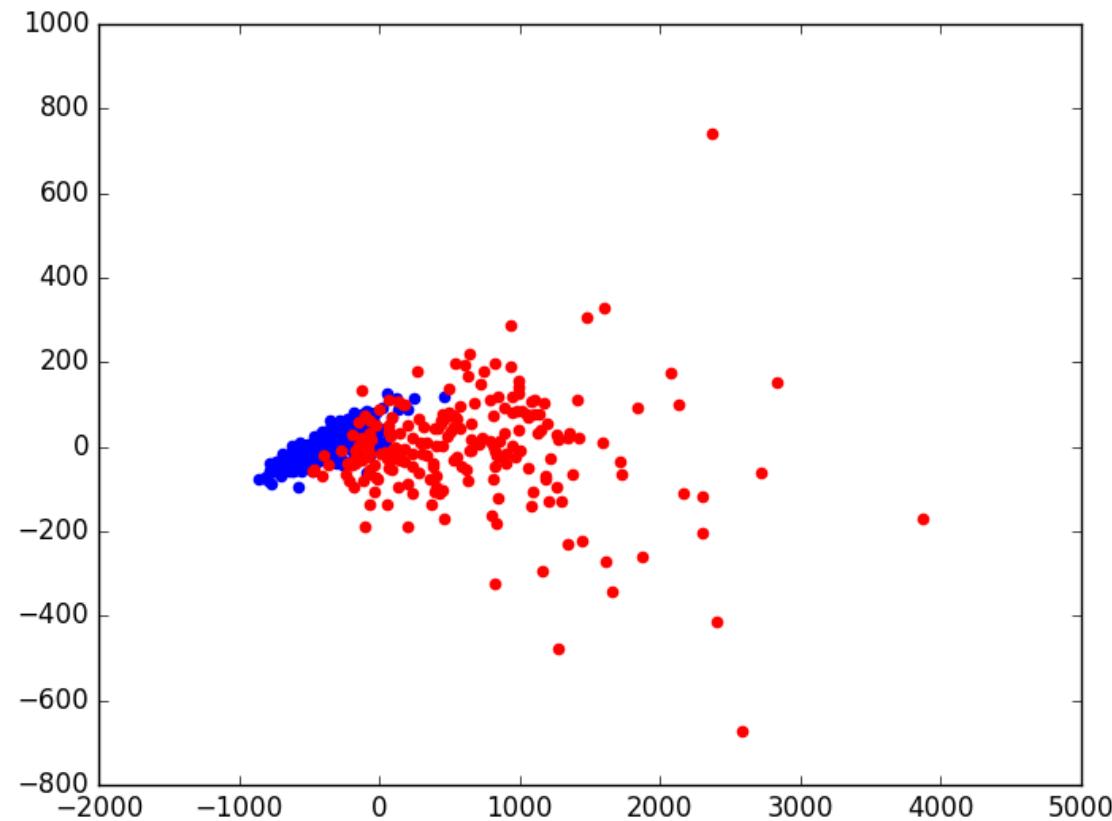
plt.show()
```

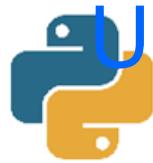


# Dimensionality reduction and visualization with PCA

---

PCA plot of breast cancer data (output of program in previous slide)





# Unsupervised learning - clustering

---

K-means: popular fast program for clustering data

Objective: find k clusters that minimize Euclidean distance of points in each cluster to their centers (means)

$$\sum_{i=1}^k \|x_j - m_i\|^2$$

$x_j \in C_i$

---



# K-means algorithm for two clusters

---

Input:  $x_i \in R^d, i = 1 \dots n$

Algorithm:

1. Initialize: assign  $x_i$  to  $C_1$  or  $C_2$  with equal probability and compute means:

2. Recompute clusters: assign  $x_i$  to  $C_1$  if  $\|x_i - m_1\| \leq \|x_i - m_2\|$ , otherwise assign to  $C_2$
3. Recompute means  $m_1$  and  $m_2$
4. Compute objective

5. Compute objective of new clustering  $\sum_{i=1}^n \|x_i - C_i\|^2$ . If difference is smaller than  $\epsilon$  then stop, otherwise go to step 2.

$\epsilon$



# K-means in scikit-learn

---

```
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import numpy as np

f = open("bc")
data = np.loadtxt(f)

X = data[:,1:]
Y = data[:,0]

clustering = KMeans(n_clusters=2,init='random').fit(X)

err = 0
for i in range(0, len(X), 1):
    if(clustering.labels_[i] != Y[i]):
        err += 1
err /= len(X)
print(err)
```

# K-means PCA plot in scikit-learn

---

```
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import numpy as np
from sklearn.decomposition import PCA

f = open("bc")
data = np.loadtxt(f)
X = data[:,1:]
Y = data[:,0]

pca = PCA(n_components=2)
pca.fit(X)
newdata = pca.transform(X)
clustering = KMeans(n_clusters=2,init='random').fit(X)

x = []
y = []
for i in range(0, len(newdata), 1):
    if(clustering.labels_[i] == 0):
        x.append(newdata[i][0])
        y.append(newdata[i][1])

plt.scatter(x, y, color='blue')

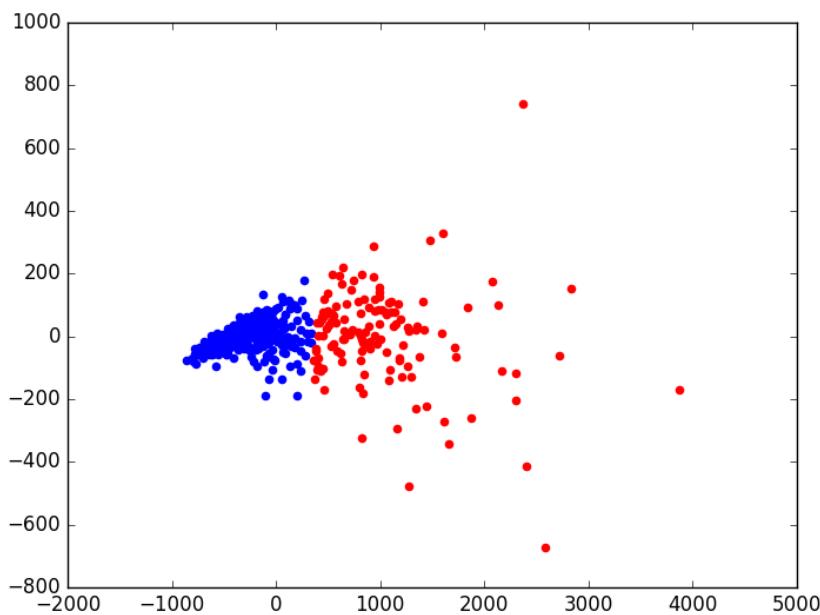
x = []
y = []
for i in range(0, len(newdata), 1):
    if(clustering.labels_[i] == 1):
        x.append(newdata[i][0])
        y.append(newdata[i][1])

plt.scatter(x, y, color='red')
plt.show()
```

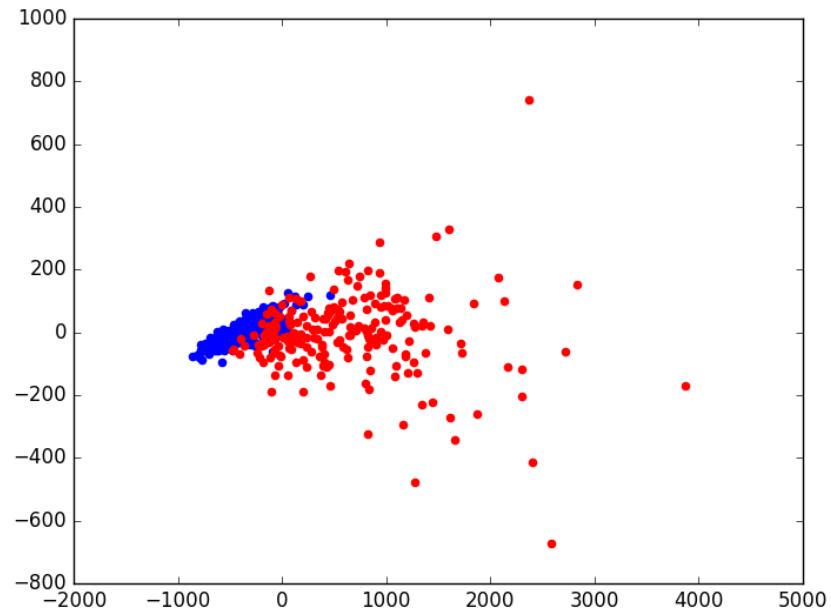
---

# K-means PCA plot in scikit-learn

PCA plot of breast cancer data  
colored by k-means labels



PCA plot of breast cancer data  
colored by true labels





# Conclusion

---

We saw basic data science and machine learning tasks in Python scikit-learn

Can we handle very large datasets in Python scikit-learn? Yes

For space use array from numpy to use a byte for a char and 4 for float and int.

Otherwise more space is used because Python is object oriented

For speed use stochastic gradient descent in scikit-learn (doesn't come with mini-batch though) and mini-batch k-means

Deep learning stuff: Keras

---



# Introducción a la regresión logística

---

MODELOS BÁSICOS



# Why use logistic regression?



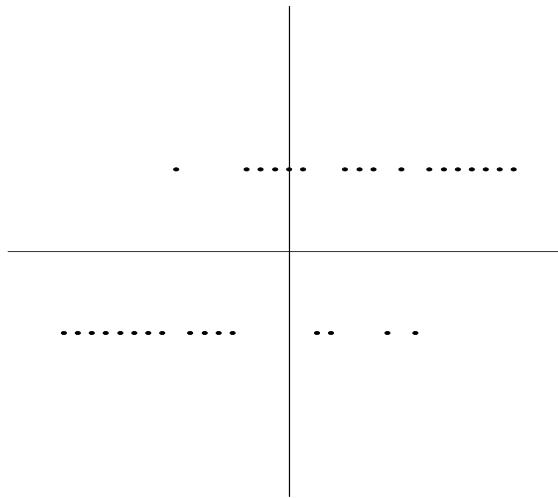
- There are many important research topics for which the dependent variable is "limited."
- For example: whether or not a person smokes, or drinks, or skips class, or takes advanced mathematics. For these the outcome is not continuous or distributed normally.
  - Example: Are mother's who have high school education less likely to have children with IEP's (individualized plans, indicating cognitive or emotional disabilities
- Binary logistic regression is a type of regression analysis where the dependent variable is a dummy variable: coded 0 (did not smoke) or 1(did smoke)



## A Problem with Linear Regression

However, transforming the independent variables does not remedy all of the potential problems. What if we have a non-normally distributed dependent variable? The following example depicts the problem of fitting a regular regression line to a non-normal dependent variable).

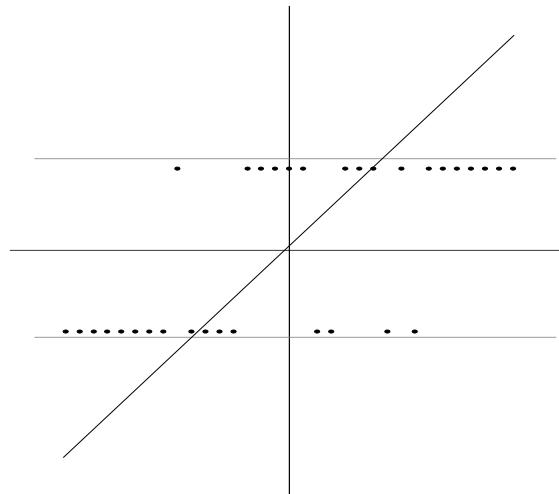
Suppose you have a binary outcome variable. The problem of having a non-continuous dependent variable becomes apparent when you create a scatterplot of the relationship. Here, we see that it is very difficult to decipher a relationship among these variables.





## A Problem with Linear Regression

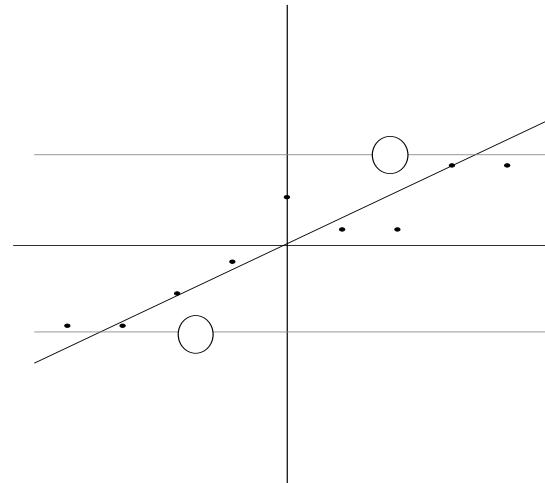
We could severely simplify the plot by drawing a line between the means for the two dependent variable levels, but this is problematic in two ways: (a) the line seems to oversimplify the relationship and (b) it gives predictions that cannot be observable values of Y for extreme values of X.



The reason this doesn't work is because the approach is analogous to fitting a linear model to the probability of the event. As you know, probabilities can only take values between 0 and 1. Hence, we need a different approach to ensure that our model is appropriate for the data.

## A Problem with Linear Regression

The mean of a binomial variable coded as (1,0) is a proportion. We could plot conditional probabilities as Y for each level of X. Of course, we could fit a linear model to these conditional probabilities, but (as shown) the linear model does not predict the maximum likelihood estimates for each group (the mean—shown by the circles) and it still produces unobservable predictions for extreme values of the dependent variable.



This plot gives us a better picture of the relationship between X and Y. It is clear that the relationship is non-linear. In fact, the shape of the curve is sigmoid.



# The Linear Probability Model

---

In the OLS regression:



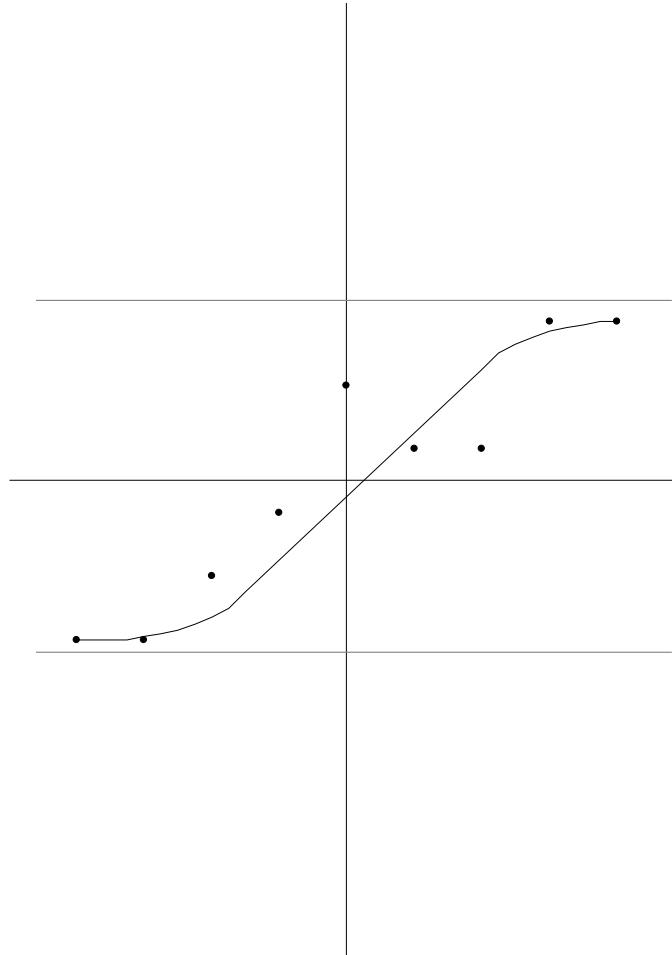
$$Y = \beta_0 + \beta_1 X + e ; \text{ where } Y = (0, 1)$$

- The error terms are heteroskedastic
  - $e$  is not normally distributed because  $Y$  takes on only two values
  - The predicted probabilities can be greater than 1 or less than 0
-



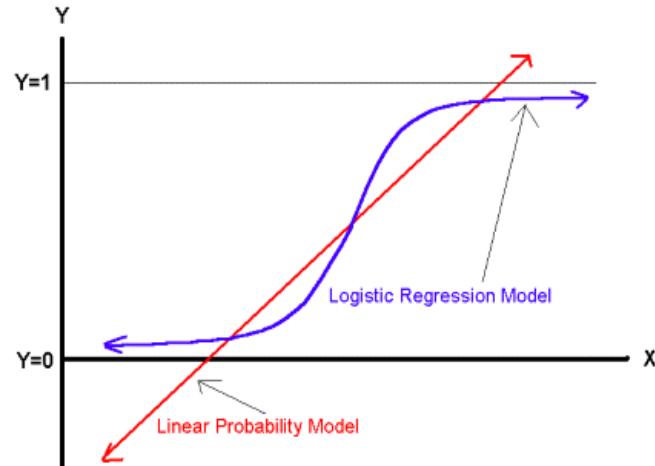
## A Problem with Linear Regression

If you think about the shape of this distribution, you may posit that the function is a cumulative probability distribution. As stated previously, we can model the nonlinear relationship between X and Y by transforming one of the variables. Two common transformations that result in sigmoid functions are **probit** and **logit** transformations. In short, a probit transformation imposes a cumulative normal function on the data. But, probit functions are difficult to work with because they require integration. Logit transformations, on the other hand, give nearly identical values as a probit function, but they are much easier to work with because the function can be simplified to a linear equation.



$$P(y|x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

### Comparing the LP and Logit Models





# The Logistic Regression Model

---

The "logit" model solves these problems:

$$\ln[p/(1-p)] = \beta_0 + \beta_1 X$$

- p is the probability that the event Y occurs,  $p(Y=1)$ 
  - [range=0 to 1]
- $p/(1-p)$  is the "odds ratio"
  - [range=0 to  $\infty$ ]
- $\ln[p/(1-p)]$ : log odds ratio, or "logit"
  - [range= $-\infty$  to  $+\infty$ ]





## Odds & Odds Ratios

Recall the definitions of an **odds**:

$$odds = \frac{p}{1-p}$$

The odds has a range of 0 to  $\infty$  with values greater than 1 associated with an event being more likely to occur than to not occur and values less than 1 associated with an event that is less likely to occur than not occur.

The **logit** is defined as the log of the odds:

$$\ln(odds) = \ln\left(\frac{p}{1-p}\right) = \ln(p) - \ln(1-p)$$

This transformation is useful because it creates a variable with a range from  $-\infty$  to  $+\infty$ . Hence, this transformation solves the problem we encountered in fitting a linear model to probabilities. Because probabilities (the dependent variable) only range from 0 to 1, we can get linear predictions that are outside of this range. If we transform our probabilities to logits, then we do not have this problem because the range of the logit is not restricted. In addition, the interpretation of logits is simple—take the exponential of the logit and you have the odds for the two groups in question.



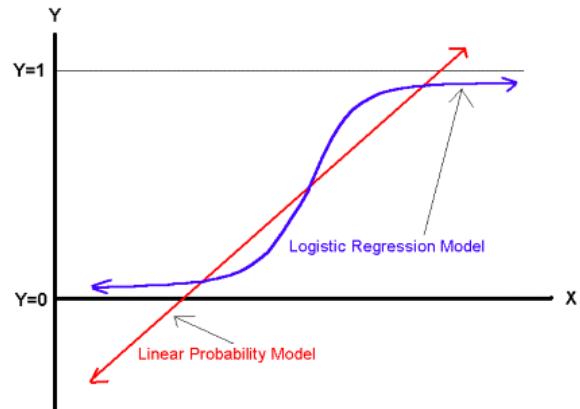
# Interpretation of Ogive

- The logistic distribution constrains the estimated probabilities to lie between 0 and 1.
- The estimated probability is:

$$p = 1/[1 + e^{(\beta_0 + \beta_1 X)}]$$

- if you let  $\beta_0 + \beta_1 X = 0$ , then  $p = .50$
- as  $\beta_0 + \beta_1 X$  gets really big,  $p$  approaches 1
- as  $\beta_0 + \beta_1 X$  gets really small,  $p$  approaches 0

### Comparing the LP and Logit Models



$$P(y|x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

# Introducing the Odds Ratio for the Logistic Transformation

If there is a 75% chance that it will rain tomorrow, then 3 out of 4 times we say this it will rain. That means for every three times it rains once it will not. The odds of it raining tomorrow are 3 to 1. This can also be understood as  $(\frac{3}{4})/\frac{1}{4}=3/1$ .

If the odds that my pony will win the race is 1 to 3, that means for every 4 races it runs, it will win 1 and lose 3. Therefore I should be paid \$3 for every dollar I bet.





## Example Interpretation of coefficient $\beta_1$

child has iep or not \* mom has high school or more Crosstabulation

		Count		Total	
		mom has high school or more			
		.00	1.00		
child has iep or not	.00	508	590	1098	
	1.00	45	33	78	
Total		553	623	1176	

$$p/(1-p)=\text{odds} \quad 5\% / 95\% = .5/.95 = .056$$

$$\text{Odds in IEP in with HS} = (33/623)/(590/623) = 33/590 = .056$$

$$8\% / 92\% = .8/.92 = .089$$

$$\text{Odds in IEP, No HS} = (45/553)/(508/553) = 45/508 = .089$$

$$\text{Change in odds due to HS} = .056/.089 = .63$$

The odds that the child of a mother with high school education has an IEP is .63 that of other mothers – it is lower because they are less likely.

Logistic regression coefficient =  $\ln(.63) = -.46$

$$\text{Change in odds} = e^{\beta_0 + \beta_1} / e^{\beta_0} = e^{\beta_1} e^{-46} = .63$$



# Hypothesis Testing

---



- The Wald statistic for the  $\beta$  coefficient is:

$$\text{Wald} = [\beta / \text{s.e.}_B]^2$$

which is distributed chi-square with 1 degree of freedom.



# Logistic Regression Reflection

---

What part is most confusing to you?

What are the possible interpretations for the part that is confusing?

Find a partner or two and share your questions





# Gradient Boosted Trees

---

MODELOS BÁSICOS



# Boosted Decision Trees



Decision Tree: Sequential application of cuts splits the data into nodes, where the final nodes (leafs) classify an event as signal or background

used since a long time in general “data-mining” applications, less known in (High Energy) Physics

similar to “simple Cuts”: each leaf node is a set of cuts.  
→ many boxes in phase space attributed either to signal or backgr.

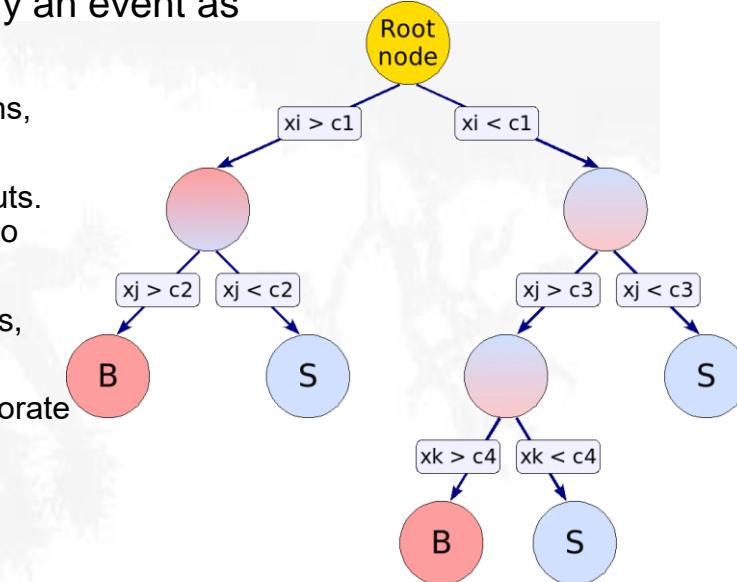
independent of monotonous variable transformations,  
immune against outliers

weak variables are ignored (and don’t (much) deteriorate performance)

Disadvantage → very sensitive to statistical fluctuations in training data

Boosted Decision Trees (1996): combine a whole forest of Decision Trees, derived from the same sample, e.g. using different event weights.

overcomes the stability problem



→ became popular in HEP since MiniBooNE, B.Roe et.a., NIM 543(2005)



# Growing a Decision Tree



start with training sample at the root node

split training sample at node into two, using a cut in  
the variable that gives best separation gain

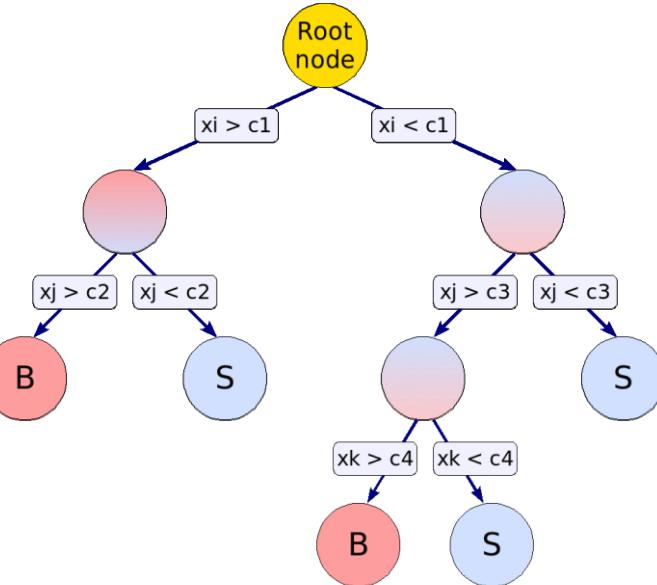
continue splitting until:

### minimal #events per node

### maximum number of nodes

maximum depth specified

~~a split doesn't give a minimum separation gain~~



leaf-nodes classify S,B according to the majority of events or give a S/B probability

Why no multiple branches (splits) per node ?

- Fragments data too quickly; also: multiple splits per node = series of binary node splits

## What about multivariate splits?

- time consuming
  - other methods more adapted for such correlations
  - we'll see later that for "boosted" DTs weak (dull) classifiers are often better, anyway



# Separation Gain



What do we mean by “best separation gain”?

define a measure on how mixed S and B are in a node:

MisClassification:

$$1 - \max(p, 1-p)$$

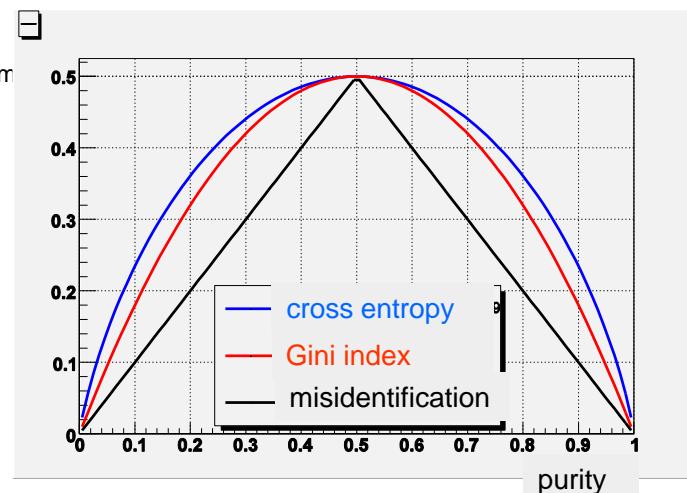
Gini-index: (Corrado Gini 1912, typically used to measure purity)

$$p(1-p) : p = \text{purity}$$

Cross Entropy:

$$-(p \ln p + (1-p) \ln(1-p))$$

difference in the various indices are small,  
most commonly used: Gini-index



separation gain: e.g.  $N_{\text{Parent}} * \text{Gini}_{\text{Parent}} - N_{\text{left}} * \text{Gini}_{\text{LeftNode}} - N_{\text{right}} * \text{Gini}_{\text{RightNode}}$

Consider all variables and all possible cut values

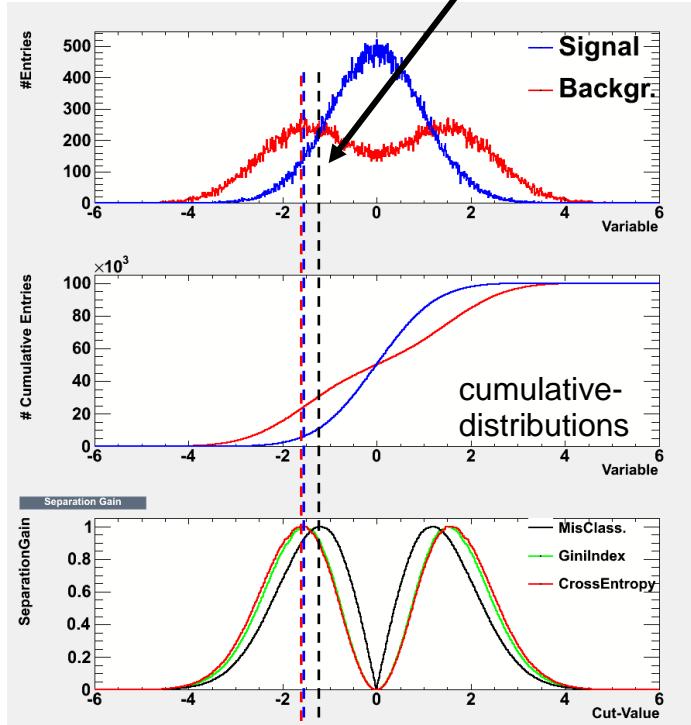
→ select variable and cut that maximises the separation gain.



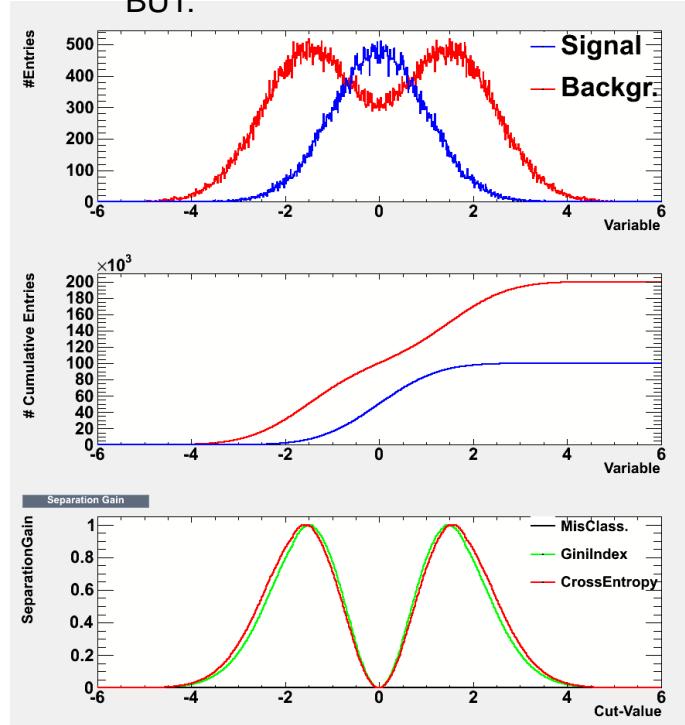
# Separation Gain



MisClassificationError → sort of the classical way to choose cut



BUT:



There are cases where the simple “misclassification” does not have any optimum at all!

other example:  $S=400, B=400 \rightarrow (S=300, B=100)$  ( $S=100, B=300$ ) or ( $S=200, B=0$ ) ( $S=200, B=400$ )  
→ equal in terms of misclassification error, but GiniIndex/Entropy favour the latter



# Decision Tree Pruning



One can continue node splitting until all leaf nodes are basically pure (using the training sample)

→ obviously: that's overtraining

Two possibilities:

stop growing earlier

generally not a good idea, even useless splits might open up subsequent useful splits

grow tree to the end and "cut back", nodes that seem statistically dominated:

→ pruning

e.g. Cost Complexity pruning:

assign to every sub-tree,  $T$   $C(T, \alpha)$  :

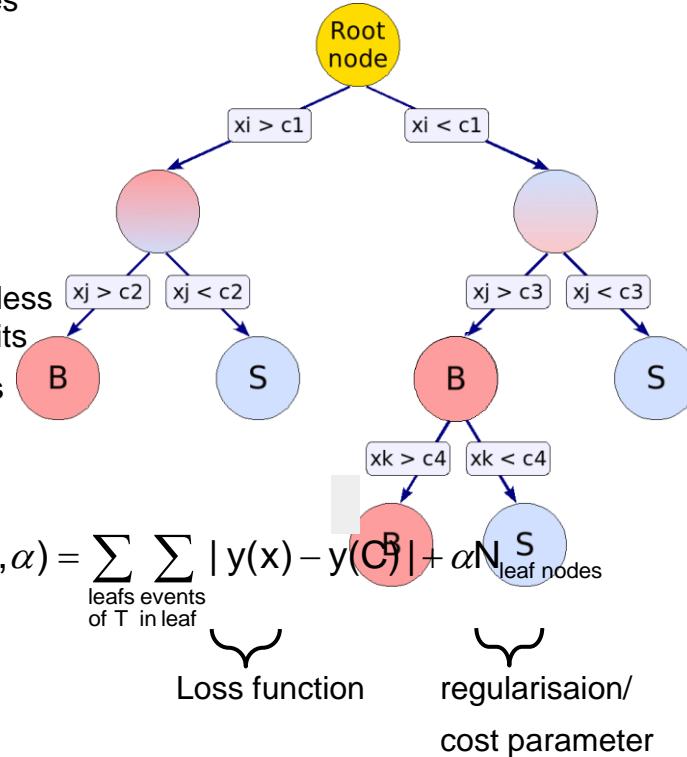
find subtree  $T$  with minimal  $C(T, \alpha)$  for given  $\alpha$

use subsequent weakest link pruning

which cost parameter  $\alpha$  ?

large enough to avoid overtraining

tuning parameter or "cross validation" (still to come in TMVA hopefully soon...)

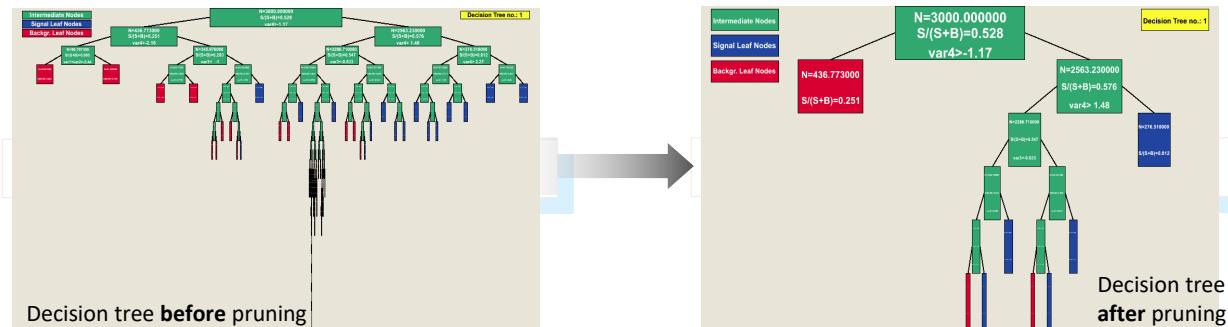




# Decision Tree Pruning



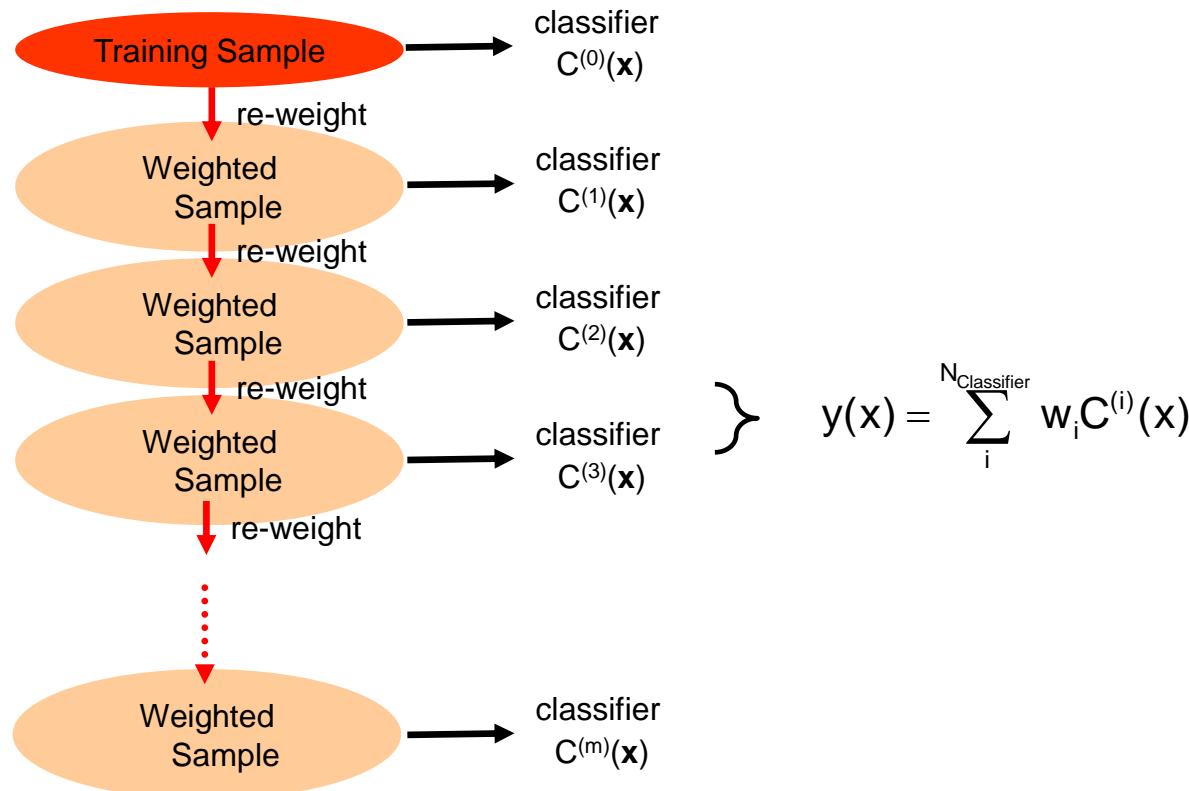
“Real life” example of an optimally pruned Decision Tree:



Pruning algorithms are developed and applied on individual trees  
optimally pruned single trees are not necessarily optimal in a forest !  
actually they tend to be TOO big when boosted, no matter how hard you prune!

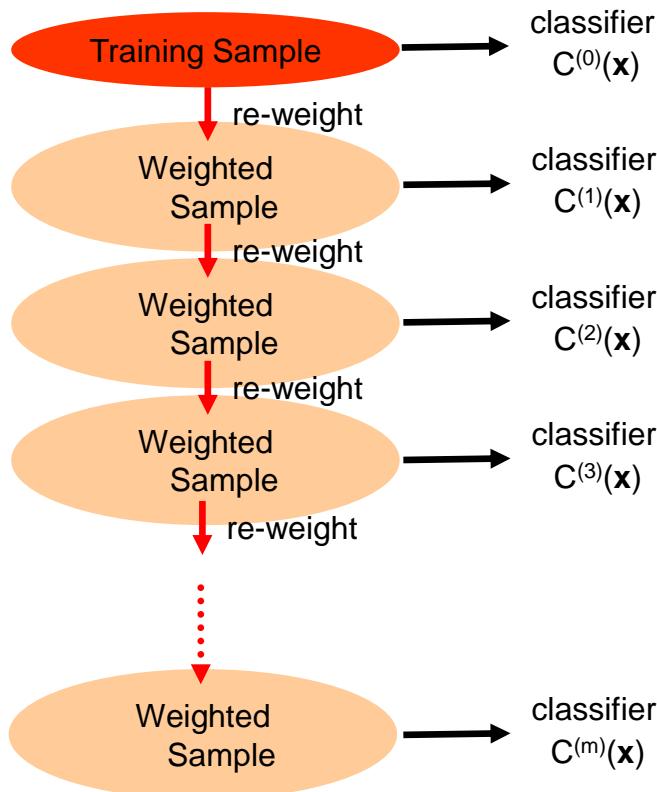


# Boosting





# Adaptive Boosting (AdaBoost)



AdaBoost re-weights events misclassified by previous classifier by:

$$\frac{1 - f_{\text{err}}}{f_{\text{err}}} \text{ with :}$$

$$f_{\text{err}} = \frac{\text{misclassified events}}{\text{all events}}$$

AdaBoost weights the classifiers also using the error rate of the individual classifier according to:

$$y(\mathbf{x}) = \sum_i^{N_{\text{Classifier}}} \log\left(\frac{1 - f_{\text{err}}^{(i)}}{f_{\text{err}}^{(i)}}\right) C^{(i)}(\mathbf{x})$$



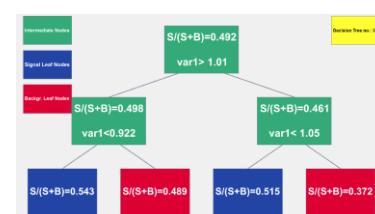
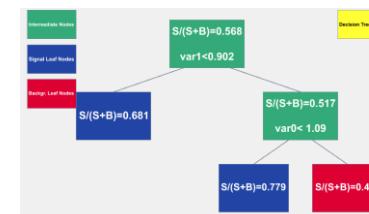
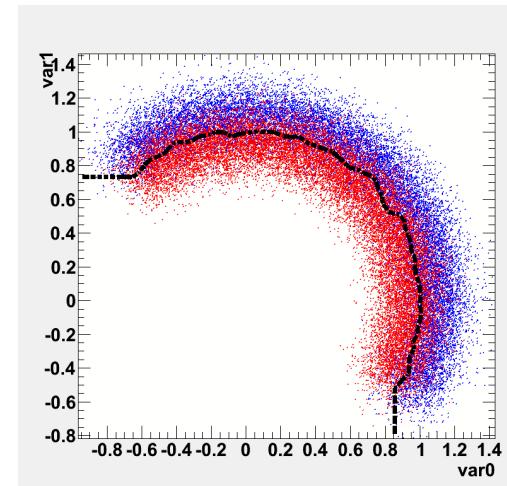
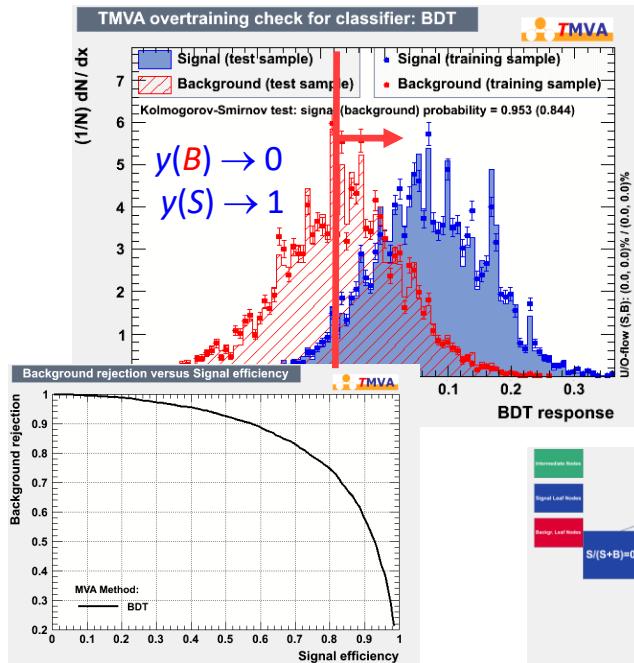
# Boosted Decision Trees



Result of ONE Decision Tree for test event is either “Signal” or “Background”

the tree gives a fixed signal eff. and background rejection

For a whole Forest however:





# AdaBoost in Pictures



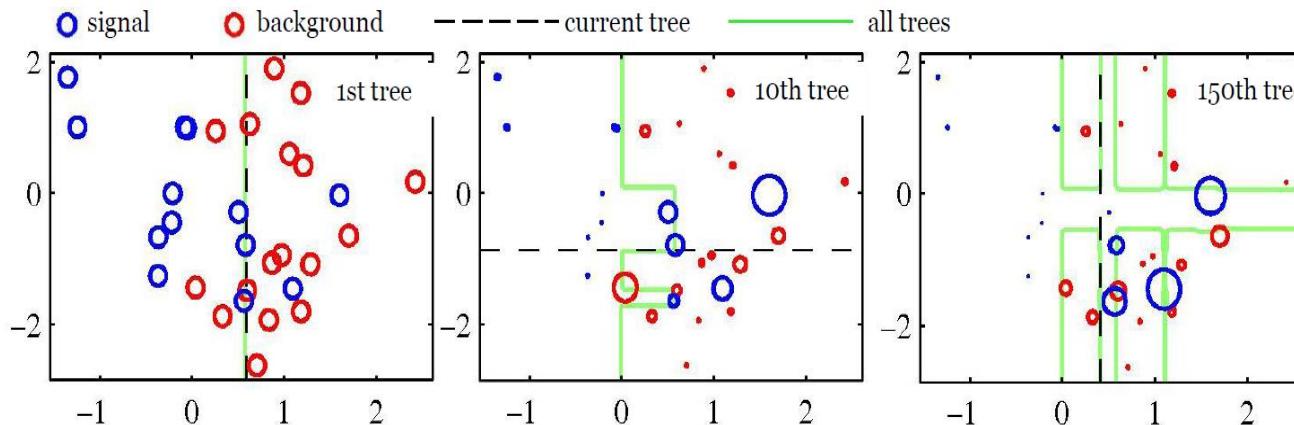
Start here:

equal event weights

misclassified events get

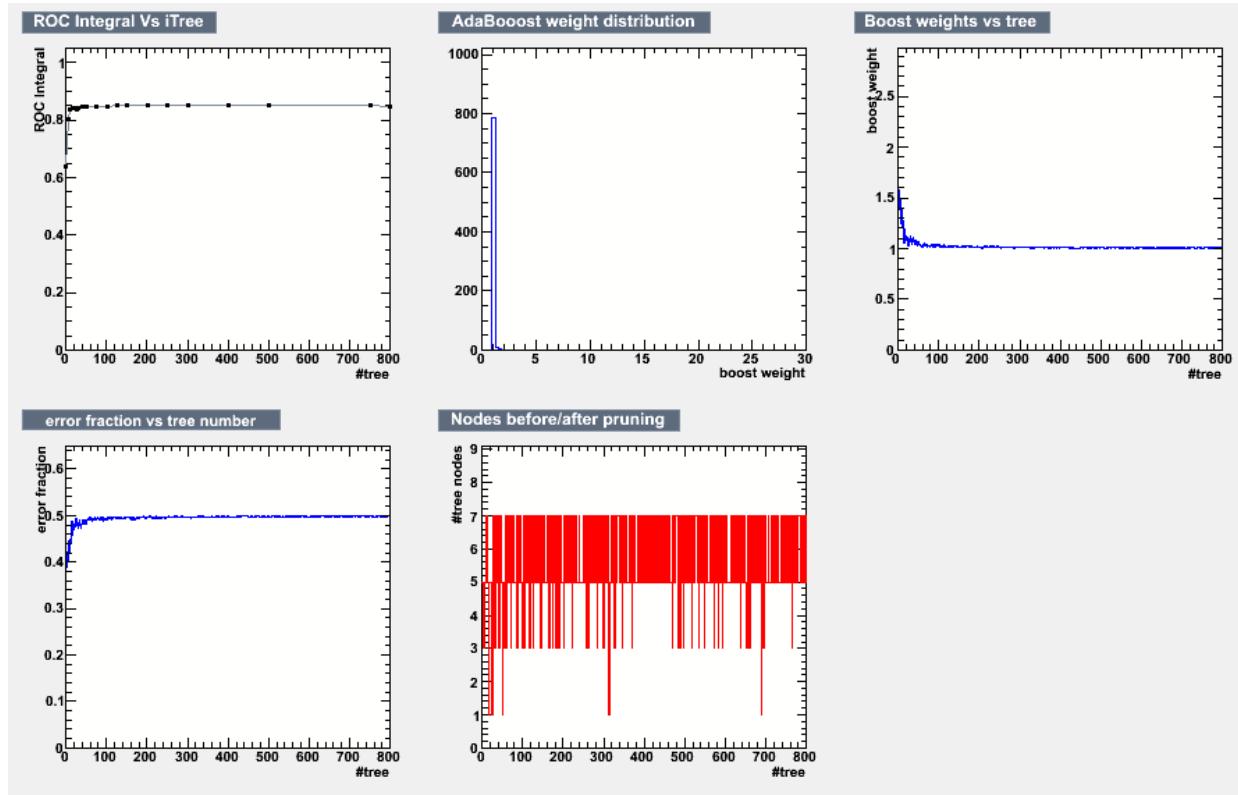
larger weights

... and so on



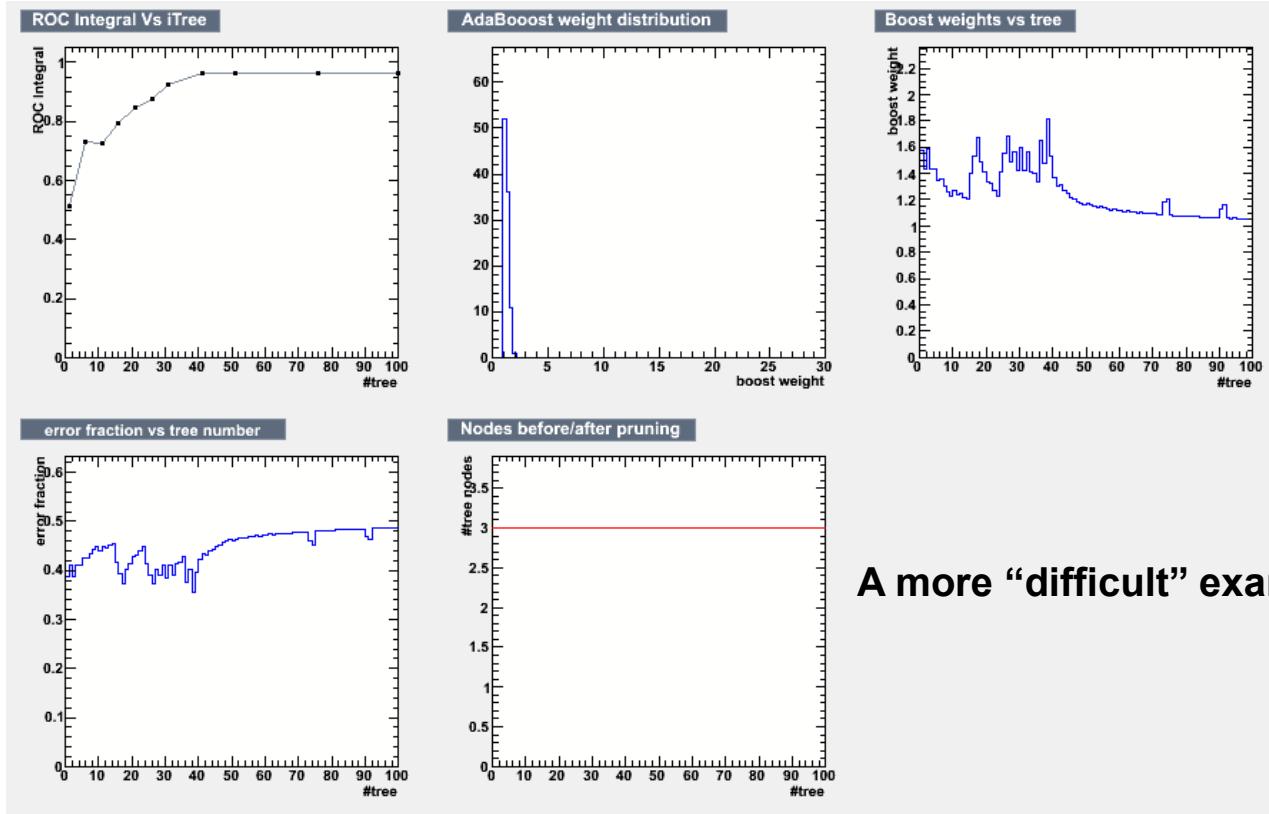


# Boosted Decision Trees – Control Plots





# Boosted Decision Trees – Control Plots



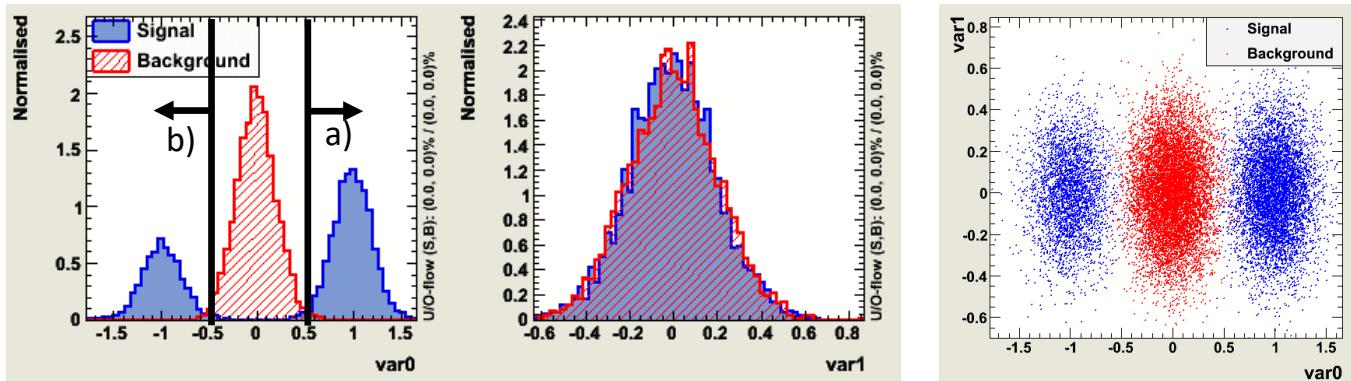
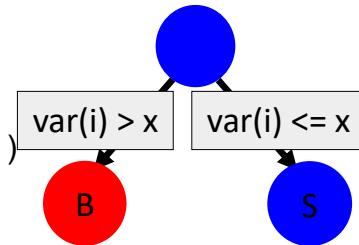
A more “difficult” example



# AdaBoost: A simple demonstration

The example: (somewhat artificial...but nice for demonstration) :

- Data file with three “bumps”
- Weak classifier (i.e. one single simple “cut”  $\leftrightarrow$  decision tree stumps )



Two reasonable cuts: a)  $\text{Var0} > 0.5 \rightarrow \epsilon_{\text{signal}} = 66\% \quad \epsilon_{\text{bkg}} \approx 0\%$  misclassified events in total 16.5%  
or

b)  $\text{Var0} < -0.5 \rightarrow \epsilon_{\text{signal}} = 33\% \quad \epsilon_{\text{bkg}} \approx 0\%$  misclassified events in total 33%  
the training of a single decision tree stump will find “cut a)”

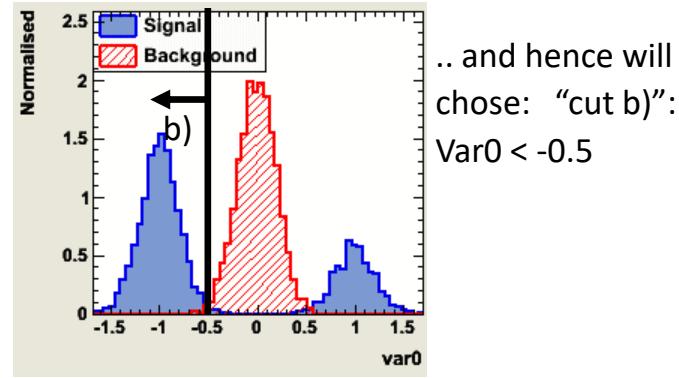
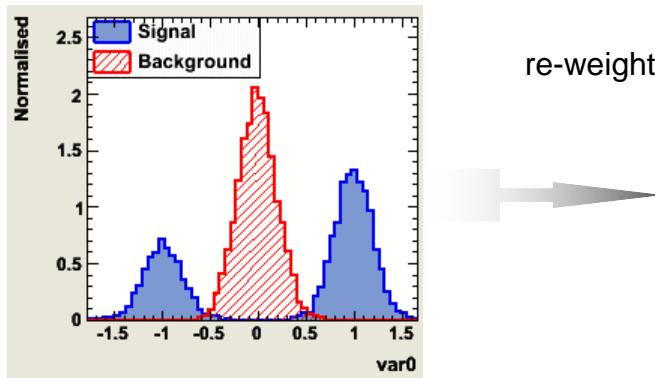


# AdaBoost: A simple demonstration

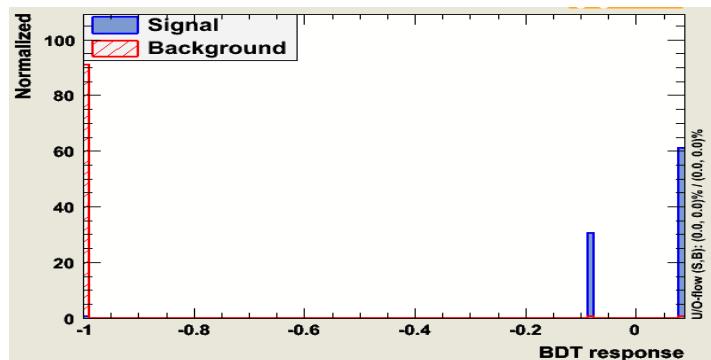


The first “tree”, choosing cut a) will give an error fraction:  $\text{err} = 0.165$

- before building the next “tree”: weight wrong classified training events by  $(1-\text{err}/\text{err}) \approx 5$
- the next “tree” sees essentially the following data sample:



The combined classifier: Tree1 + Tree2  
the (weighted) average of the response to a test event from both trees is able to separate signal from background as good as one would expect from the most powerful classifier

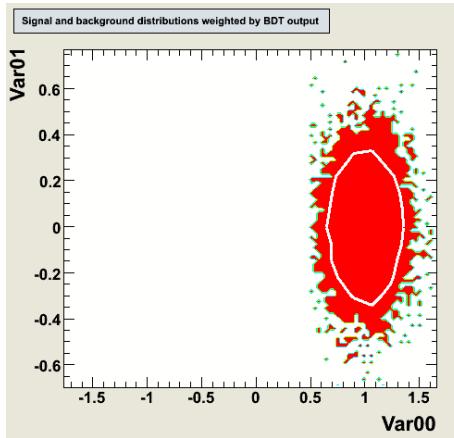




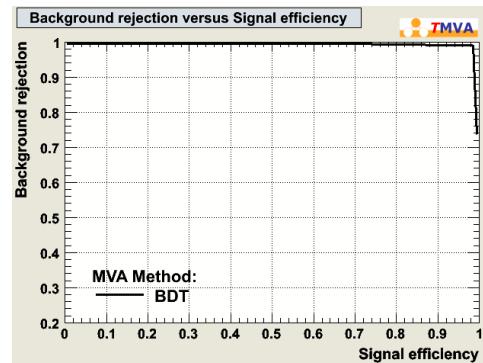
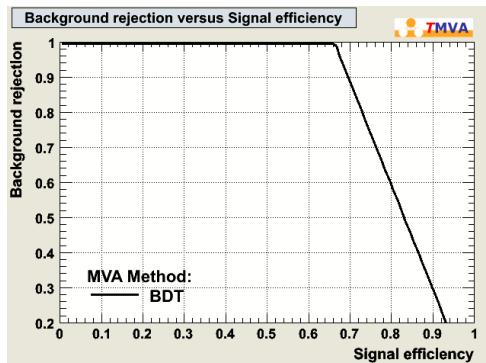
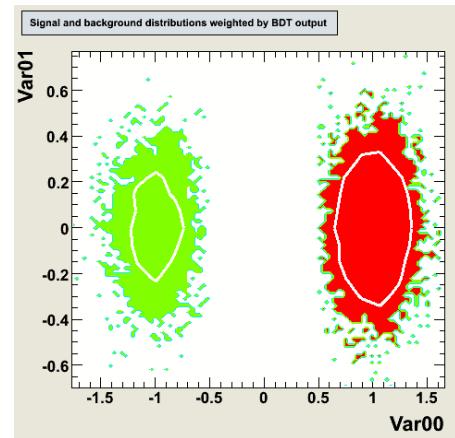
# AdaBoost: A simple demonstration



Only 1 tree “stump”



Only 2 tree “stumps” with AdaBoost





# “A Statistical View of Boosting”

(Friedman 1998 et.al)

---

## ■ Boosted Decision Trees: two different interpretations

- ▶ give events that are “difficult to categorize” more weight and average afterwards the results of all classifiers that were obtained with different weights

- ▶ see each Tree as a “basis function” of a possible classifier →

- boosting or bagging is just a mean to generate a set of “basis functions”

- linear combination of basis functions gives final classifier or: final classifier is an expansion in the basis functions.

$$y(\vec{\alpha}, x) = \sum_{\text{tree}} \alpha_i T_i(x)$$

- every “boosting” algorithm can be interpreted as optimising the loss function in a “greedy stagewise” manner

- i.e. from the current point in the optimisation – e.g. *building of the decision tree forest*- :

- chooses the parameters for the next boost step (weights) such that one moves along the steepest gradient of the loss function

- AdaBoost: “exponential loss function” =  $\exp(-y_0 y(\alpha, x))$  where  $y_0 = -1$  (bkg),  $y_0 = 1$  (signal)



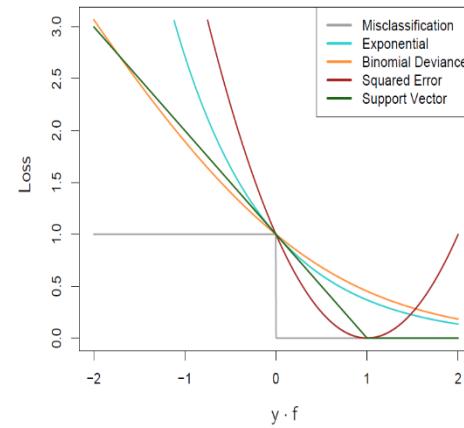
# Gradient Boost



Gradient Boost is a way to implement “boosting” with arbitrary “loss functions” by approximating “somehow” the gradient of the loss function

AdaBoost: Exponential loss  $\exp(-y_0 y(\alpha, x)) \rightarrow$  theoretically sensitive to outliers

Binomial log-likelihood loss  $\ln(1 + \exp(-2y_0 y(\alpha, x))) \rightarrow$  more well behaved loss function, (the corresponding “GradientBoost” is implemented in TMVA)





# Bagging and Randomised Trees

---

other classifier combinations:

Bagging:

combine trees grown from “bootstrap” samples  
(i.e re-sample training data with replacement)

Randomised Trees: (**Random Forest: trademark L.Breiman, A.Cutler**)

combine trees grown with:

random bootstrap (or subsets) of the training data only  
consider at each node only a random subsets of variables for the split  
NO Pruning!

These combined classifiers work surprisingly well, are very stable and almost perfect “out of the box” classifiers

---



# AdaBoost vs Bagging and Randomised Forests

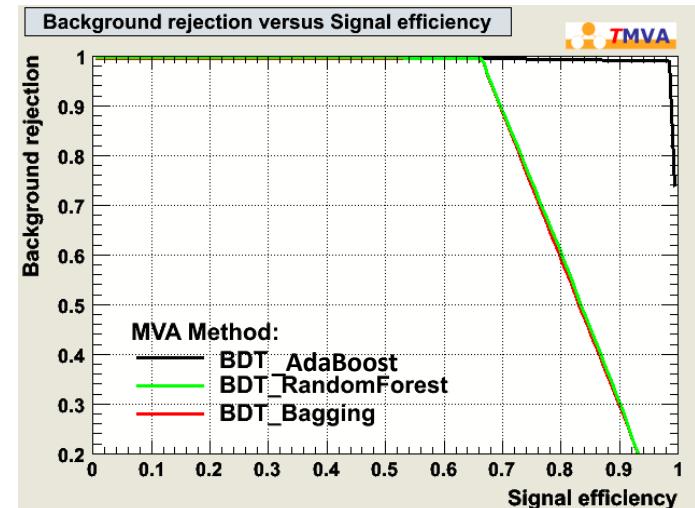
Sometimes people present “boosting” as nothing else then just “smearing” in order to make the Decision Trees more stable w.r.t statistical fluctuations in the training.

→ clever “boosting” however can do more, than for example: for previous example of “three bumps”

- Random Forests
- Bagging

as in this case, pure statistical fluctuations are not enough to enhance the 2<sup>nd</sup> peak sufficiently

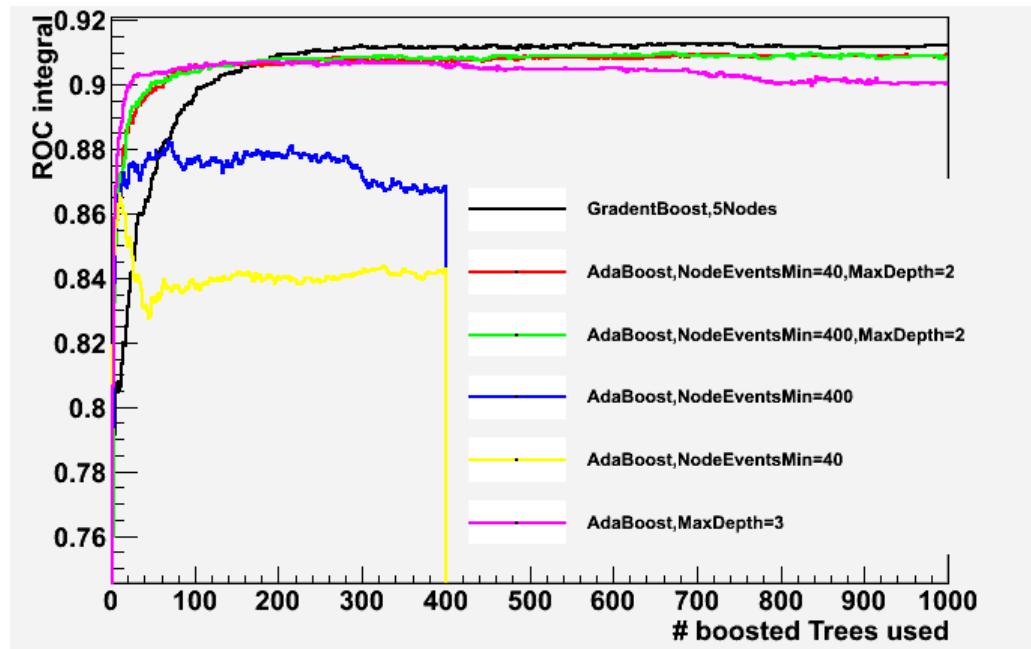
however: a “fully grown decision tree” is much more than a “weak classifier”  
→ “stabilization” aspect is more important



Surprisingly: Often using smaller trees (weaker classifiers) in AdaBoost and other clever boosting algorithms (i.e. gradient boost) seems to give overall significantly better performance !



# Boosting at Work



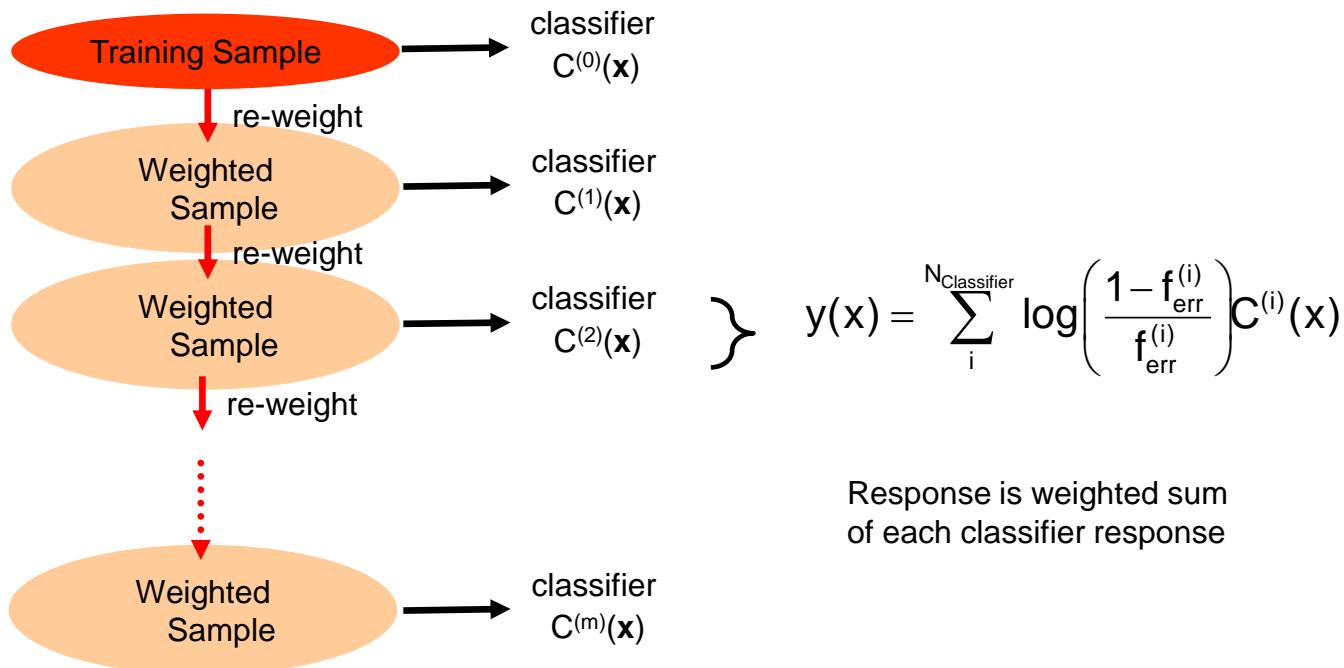
Boosting seems to work best on “weak” classifiers (i.e. small, dum trees)

Tuning (tree building) parameter settings are important

For good out of the box performance: Large numbers of very small trees

# Generalised Classifier Boosting

- Principle (just as in BDT): multiple training cycles, each time wrongly classified events get a higher event weight



Boosting might be interesting especially for simple (weak) Methods like Cuts, Linear Discriminants, simple (small, few nodes) MLPs

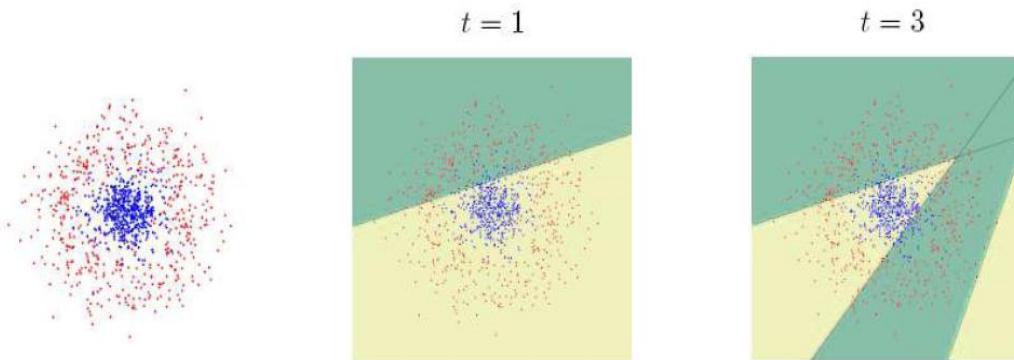


# AdaBoost On a linear Classifier (e.g. Fisher)

---

J. Sochman, J. Matas, [cmp.felk.cvut.cz](http://cmp.felk.cvut.cz)

Start with a problem for which a linear classifier is weak:

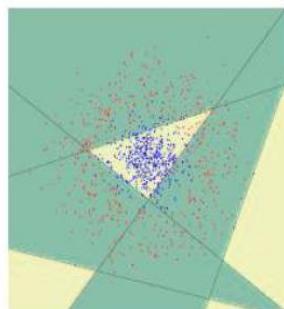




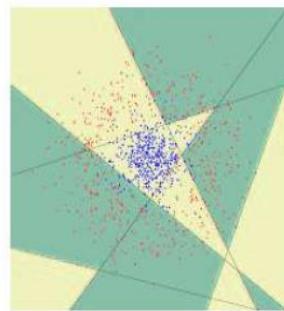
# AdaBoost On a linear Classifier (e.g. Fisher)



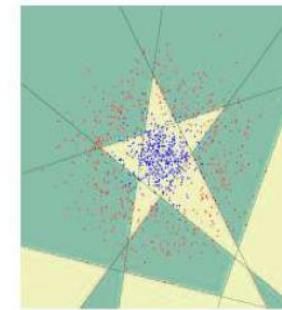
$t = 5$



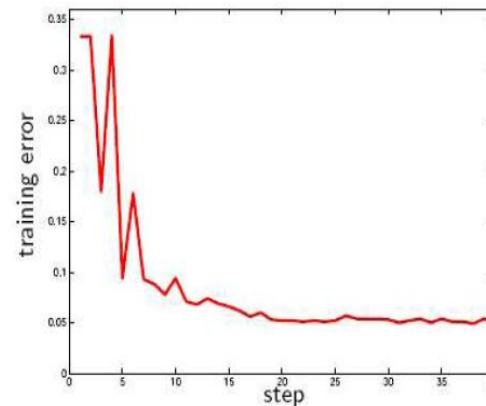
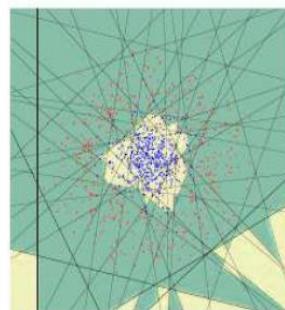
$t = 6$



$t = 7$



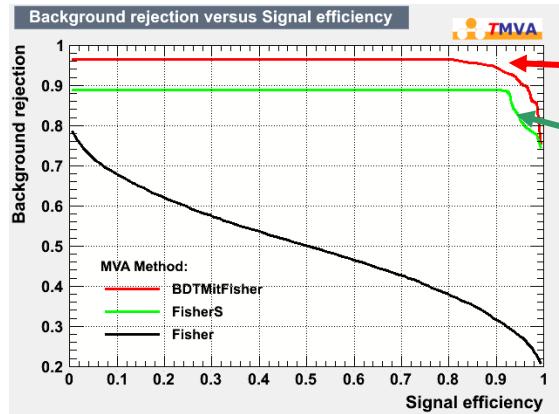
$t = 40$



Ups... there's still a problem in TMVA's generalized boosting. This example doesn't work yet !



# Boosting a Fisher Discriminant in TMVA...

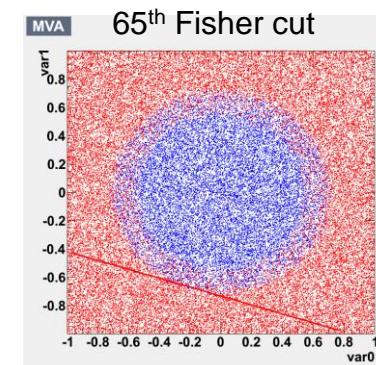
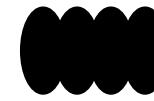
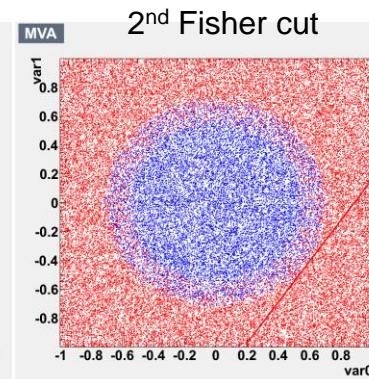
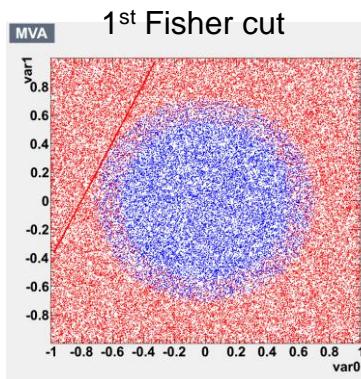


100 Boosts of a “Fisher Discriminant”

as Multivariate Tree split (yes.. it is in TMVA although I argued against it earlier. I hoped to cope better with linear correlations that way...)

generalised boosting of Fisher classifier

→ Something isn't quite correct yet !





# Learning with Rule Ensembles



Following RuleFit approach by [Friedman-Popescu](#)

Friedman-Popescu, Tech Rep,  
Stat. Dpt, Stanford U., 2003

Model is linear combination of *rules*, where a rule is a sequence of cuts (i.e. a branch of a decision tree)

The diagram shows the RuleFit classifier equation:

$$y_{RF}(\vec{x}) = a_0 + \sum_{m=1}^{M_R} a_m r_m(\vec{x}) \cdot \sum_{k=1}^{n_R} b_k \hat{x}_k$$

Annotations explain the components:

- RuleFit classifier: Points to the overall equation.
- rules (cut sequence →  $r_m=1$  if all cuts satisfied, =0 otherwise): Points to the term  $\sum_{m=1}^{M_R} a_m r_m(\vec{x})$ .
- normalised discriminating event variables: Points to the term  $\sum_{k=1}^{n_R} b_k \hat{x}_k$ .
- Sum of rules: Points to the summation index  $m$  in  $\sum_{m=1}^{M_R}$ .
- Linear Fisher term: Points to the summation index  $k$  in  $\sum_{k=1}^{n_R}$ .

The problem to solve is

Create rule ensemble: use forest of decision trees

*pruning removes topologically equal rules (same variables in cut sequence)*

Add a “Fisher term” to capture linear correlations

Fit coefficients  $a_m, b_k$ : gradient direct regularization minimising *Risk* (Friedman et al.)



# Regression Trees

---



Rather than calling leafs Signal or Background

- could also give them “values” (i.e. “mean value” of all values attributed to training events that end up in the node)
- Regression Tree

Node Splitting: Separation Gain → Gain in Variance (RMS) of target function

Boosting: error fraction → “distance” measure from the mean  
linear, square or exponential

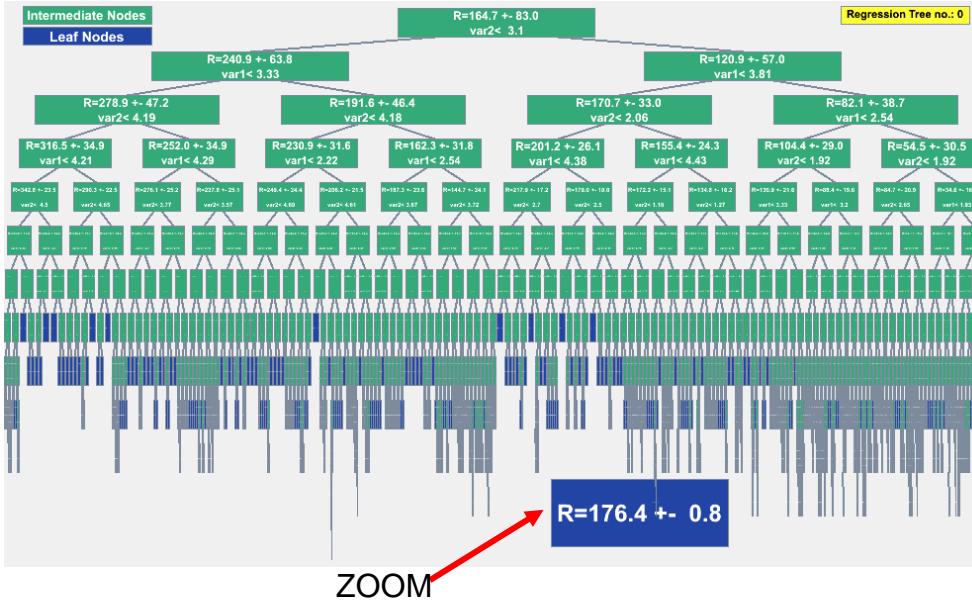
Use this to model ANY non analytic function of which you have “training data”

i.e.

energy in your calorimeter as function of show parameters  
training data from testbeam



# Regression Trees



Leaf Nodes:  
One output value

Regression Trees seem to need DESPITE BOOSTING larger trees



# Summary

---



Boosted Decision Trees → a “brute force method” works “out of the box”

check tuning parameters anyway.

start with “small trees” (limit the maximum number of splits (tree depth)

automatic tuning parameter optimisation

first implementation is done, obviously needs LOTS of time!

be as careful as with “cuts” and check against data

Boosting can (in principle) be applied to any (weak) classifier

Boosted Regression Trees → at least as much “brute force”

little experience with yet.. but probably equally robust and powerful

---

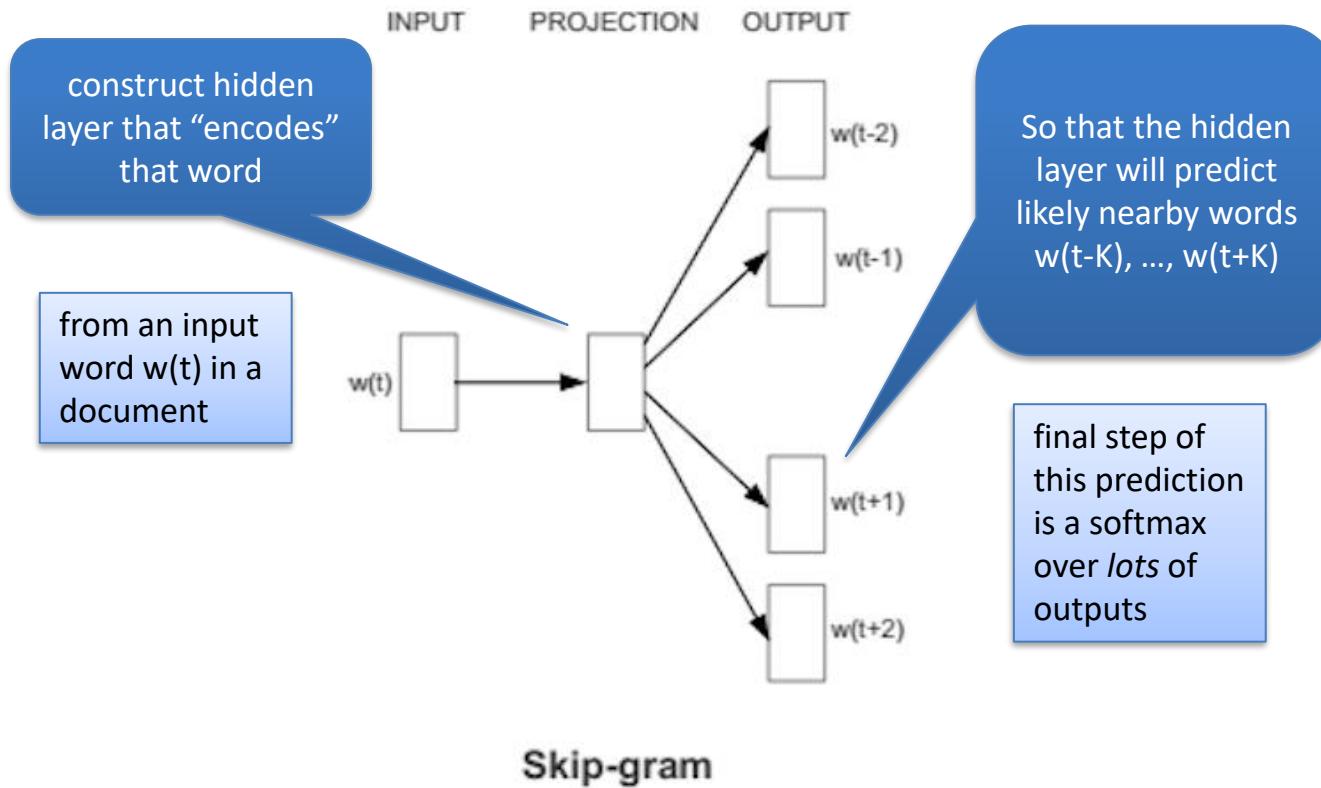


# Word2Vec and Word embeddings

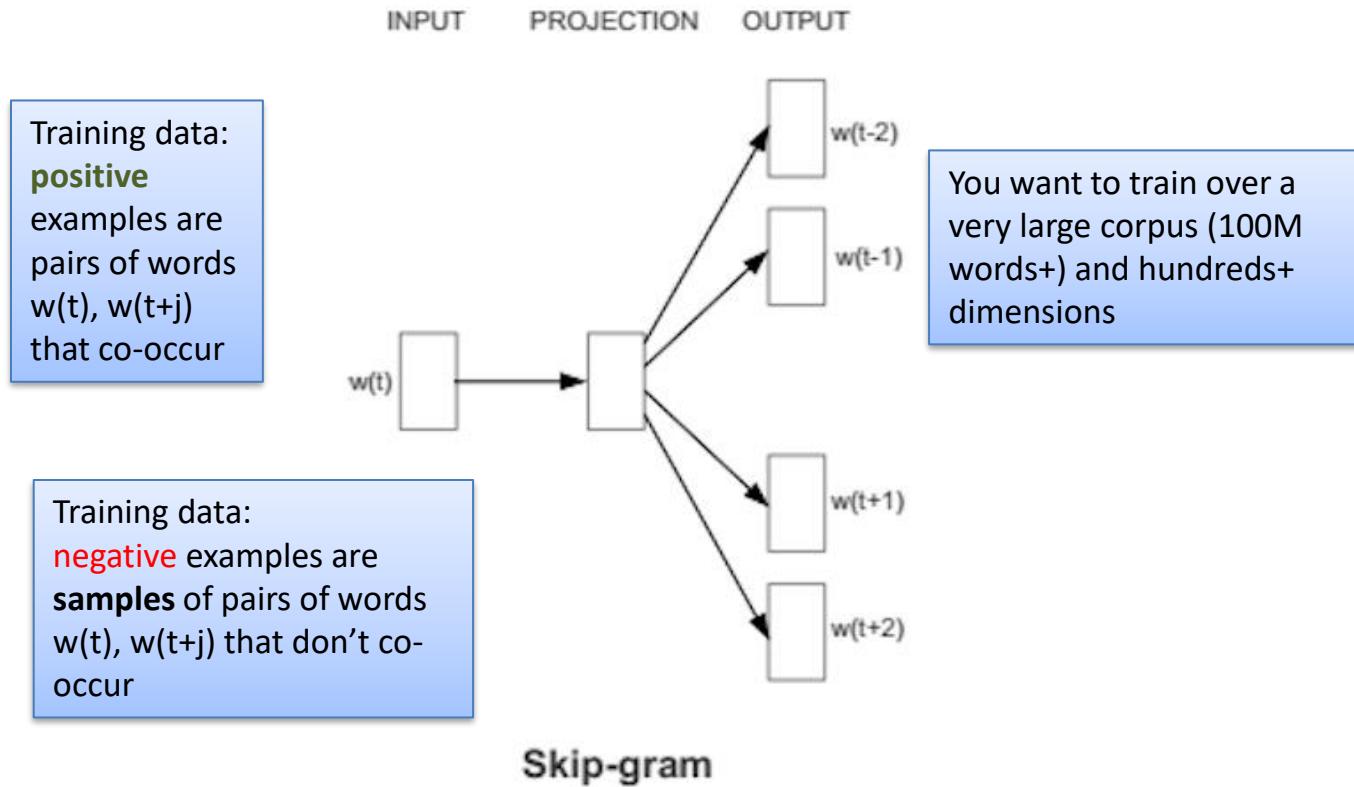
---

MODELOS BÁSICOS

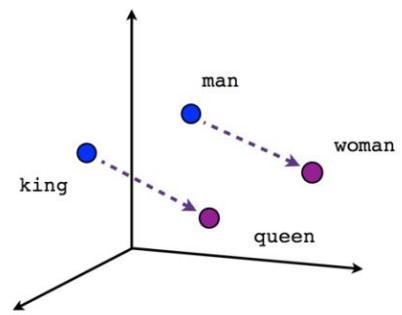
# Basic idea behind skip-gram embeddings



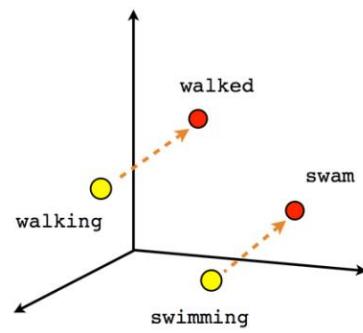
# Basic idea behind skip-gram embeddings



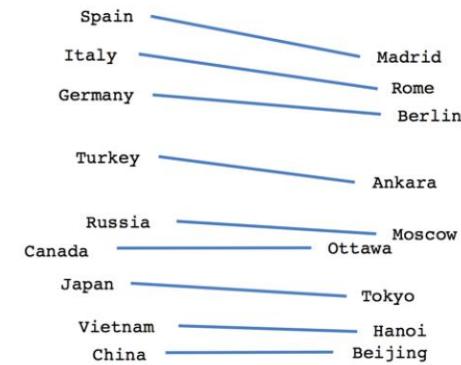
# Results from word2vec



Male-Female



Verb tense

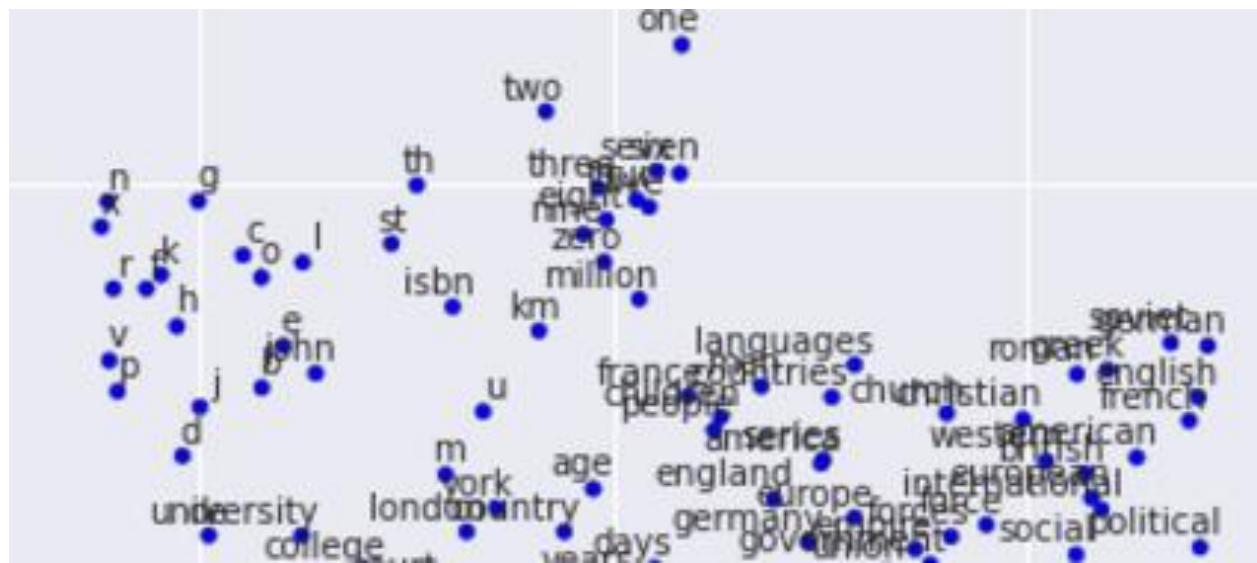


Country-Capital

<https://www.tensorflow.org/versions/r0.7/tutorials/word2vec/index.html>

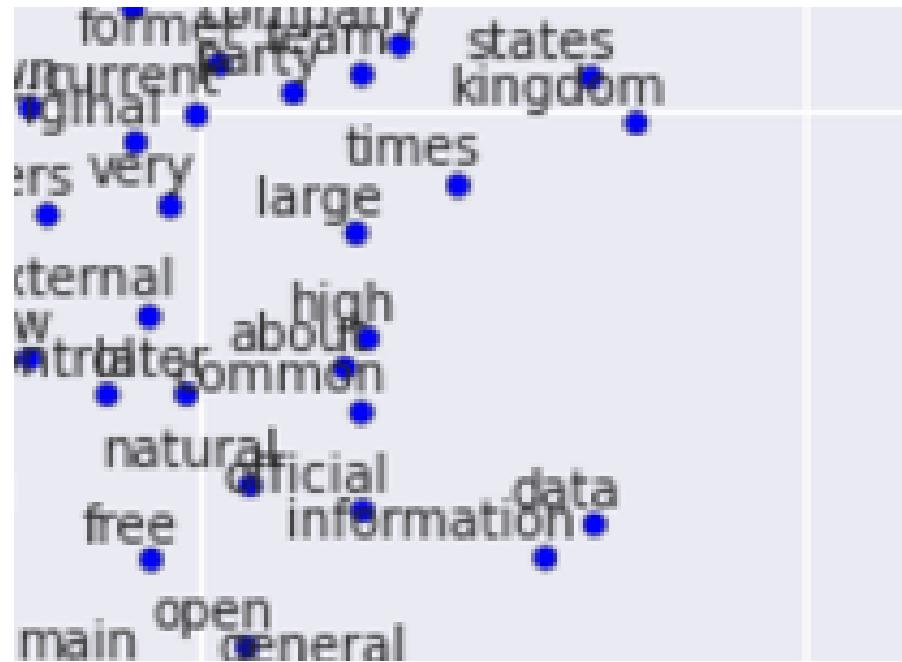


# Results from word2vec



<https://www.tensorflow.org/versions/r0.7/tutorials/word2vec/index.html>

# Results from word2vec



<https://www.tensorflow.org/versions/r0.7/tutorials/word2vec/index.html>



---

What's new in ANNs in the last 5-10 years?

**Deeper networks**, more data, and faster training

Scalability and use of GPUs ✓

Symbolic differentiation ✓

Some subtle changes to cost function, architectures,  
optimization methods ✓

What types of ANNs are most successful and why?

Convolutional networks (CNNs) ✓

Long term/short term memory networks (LSTM) ✓

Word2vec and embeddings✓

What are the **hot research topics** for deep learning?

---



# Some current hot topics

---

## Multi-task learning



Does it help to learn to predict many things at once? e.g., POS tags and NER tags in a word sequence?

Similar to word2vec learning to produce all context words

Extensions of LSTMs that model memory more generally

e.g. for question answering about a story



# Some current hot topics

---

Optimization methods (>> SGD)

Neural models that include “attention”

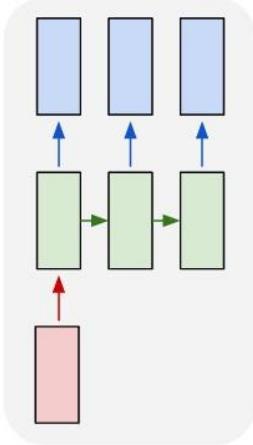
Ability to “explain” a decision



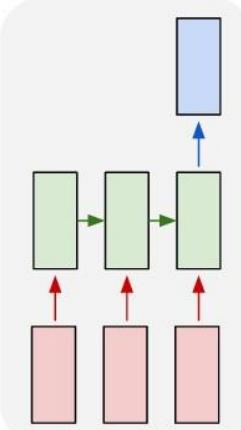
# Examples of attention



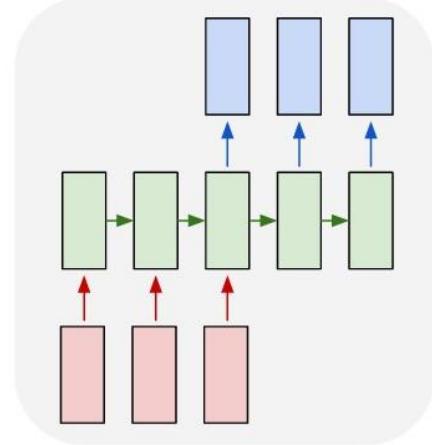
one to many



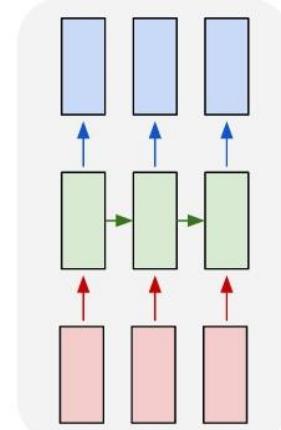
many to one



many to many

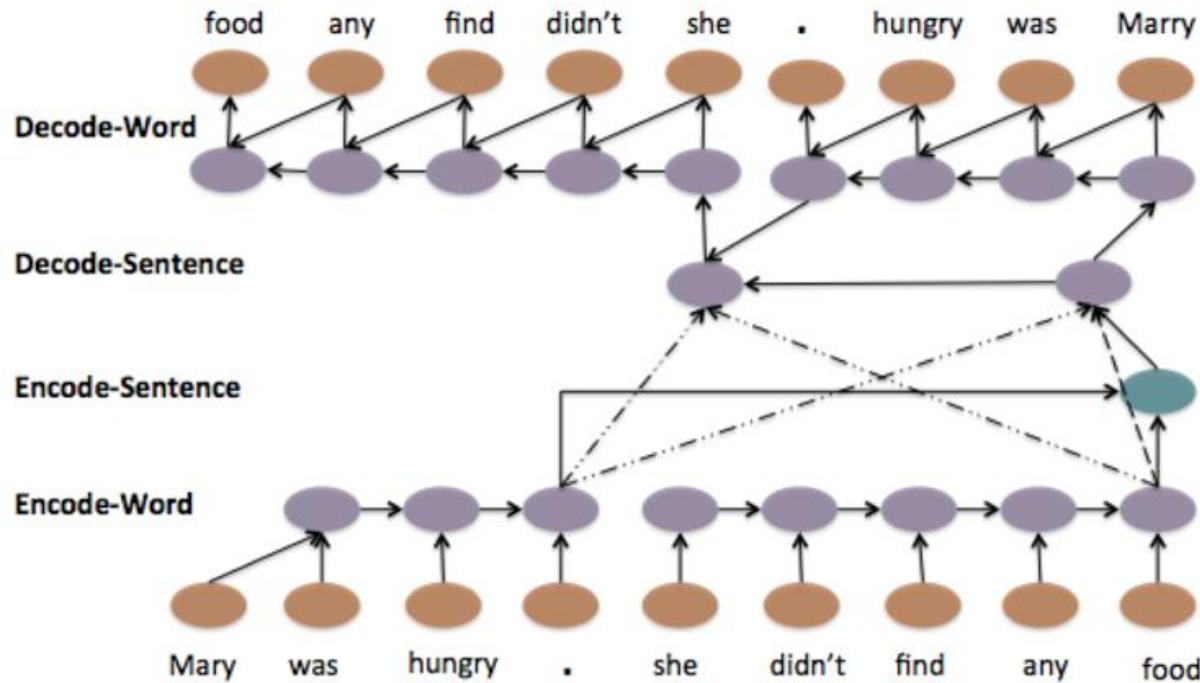


many to many



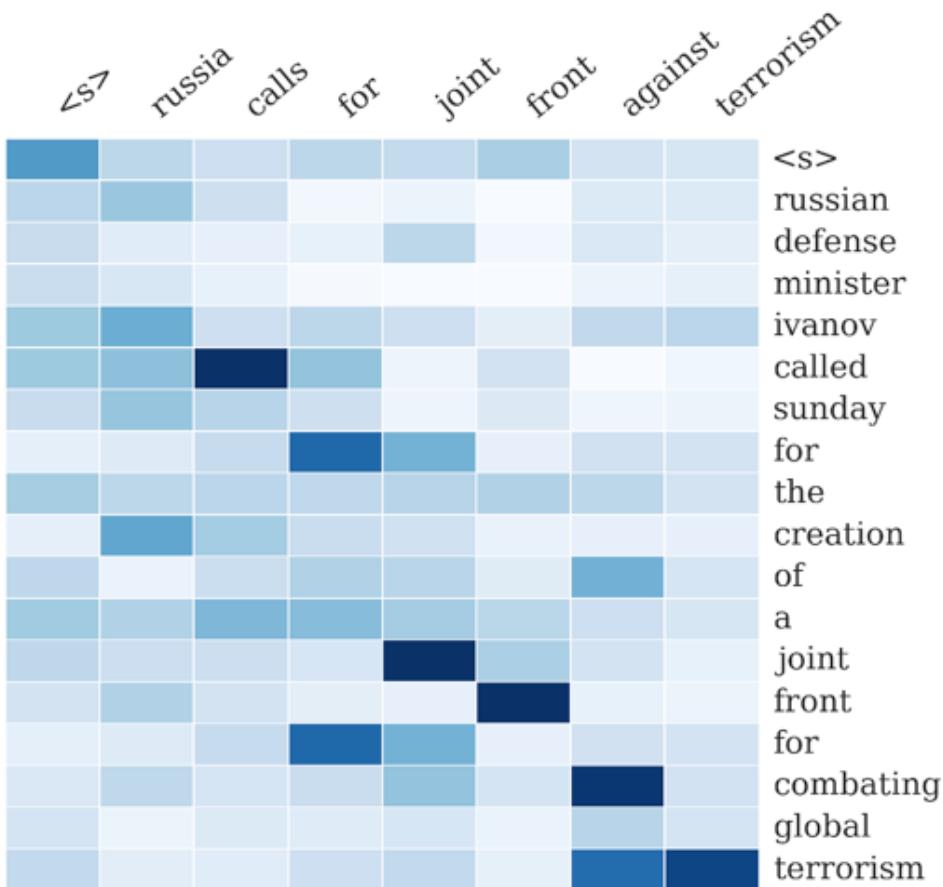
Basic idea: similarly to the way an LSTM chooses what to “forget” and “insert” into memory, allow a network to choose what inputs to “attend to” in generation phase

# Examples of attention



<http://yanran.li/peppypapers/2015/10/07/survey-attention-model-1.html>

ACL 15, Li, Luong, Jurafsky



<http://yanran.li/peppypapers/2015/10/07/survey-attention-model-1.html>  
EMNLP 15, Rush, Chopra, Weston



# Some current hot topics

---

Knowledge-base embedding: extending word2vec to embed large databases of facts about the world into a low-dimensional space.



TransE, TransR, ...

“NLP from scratch”: sequence-labeling and other NLP tasks with minimal amount of feature engineering, only networks and character- or word-level embeddings



# Some current hot topics

---

Computer vision: complex tasks like generating a natural language caption from an image or understanding a video clip

Machine translation

    English to Spanish, ...

Using neural networks to perform *tasks*

    Driving a car

    Playing games (like Go or ...)

**Reinforcement learning**





# Máquinas Soporte Vectorial

---

MODELOS BÁSICOS



# History of SVM

---



SVM is related to statistical learning theory [3]

SVM was first introduced in 1992 [1]

SVM becomes popular because of its success in handwritten digit recognition

1.1% test error rate for SVM. This is the same as the error rates of a carefully constructed neural network, LeNet 4.

See Section 5.11 in [2] or the discussion in [3] for details

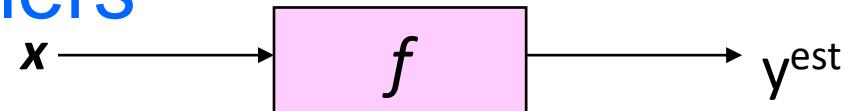
SVM is now regarded as an important example of “kernel methods”, one of the [key area in machine learning](#)

Note: the meaning of “kernel” is different from the “kernel” function for Parzen windows

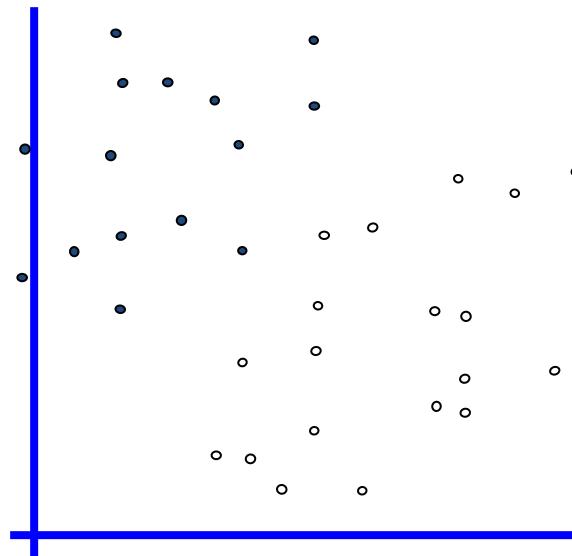


# Linear Classifiers

Estimation:



- denotes +1
- denotes -1



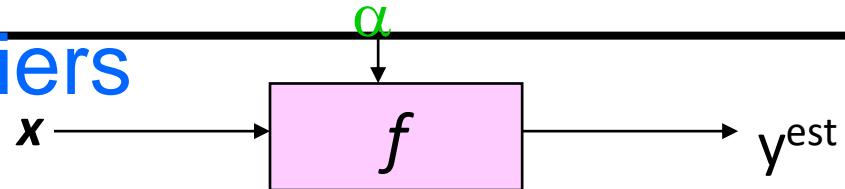
$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} - b)$$

$\mathbf{w}$ : weight vector  
 $\mathbf{x}$ : data vector

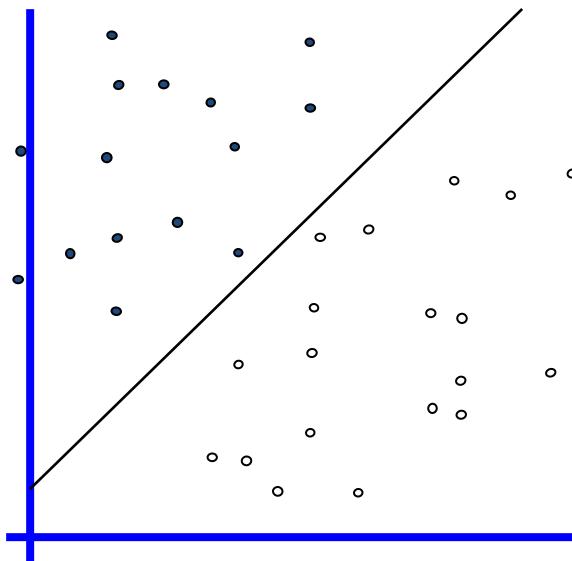
How would you  
classify this data?



# Linear Classifiers



- denotes +1
- denotes -1

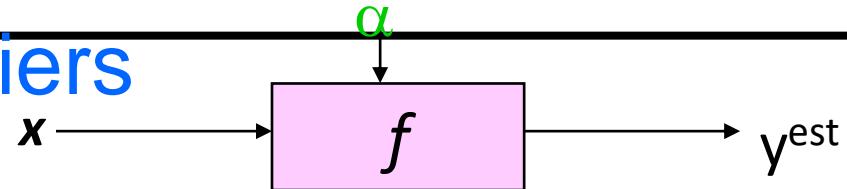


$$f(x, w, b) = \text{sign}(w \cdot x - b)$$

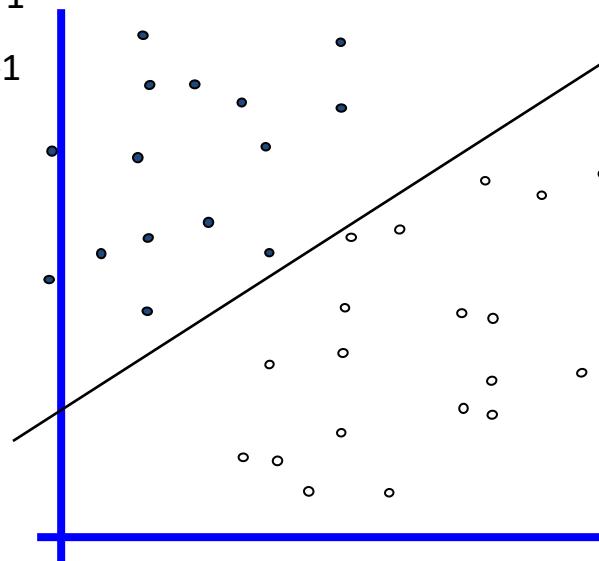
How would you  
classify this data?



# Linear Classifiers



- denotes +1
- denotes -1

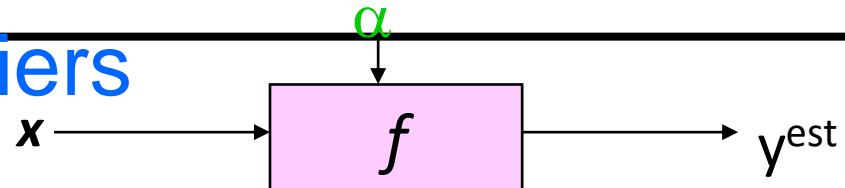


$$f(x, w, b) = \text{sign}(w \cdot x - b)$$

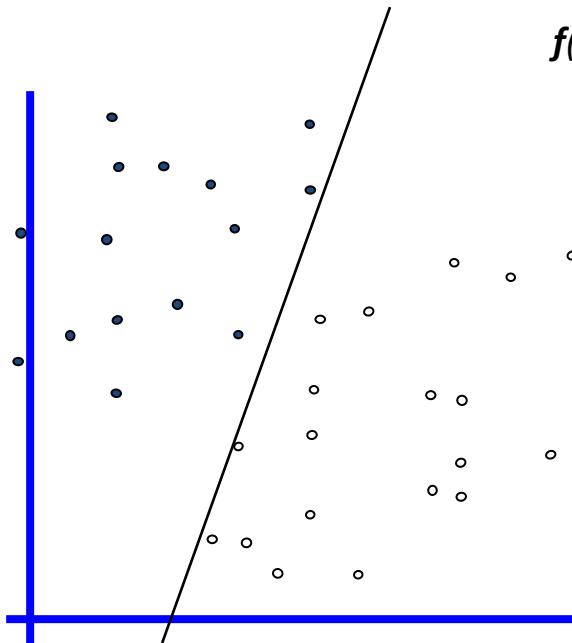
How would you  
classify this data?



# Linear Classifiers



- denotes +1
- denotes -1

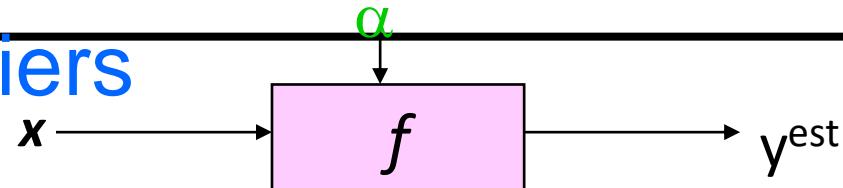


$$f(x, w, b) = \text{sign}(w \cdot x - b)$$

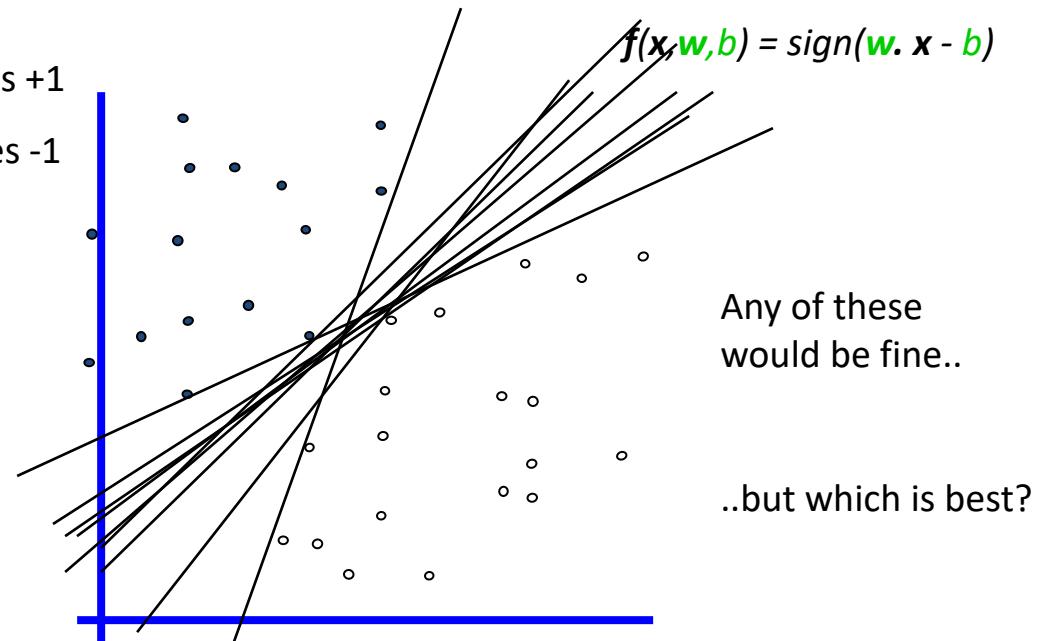
How would you  
classify this data?



# Linear Classifiers

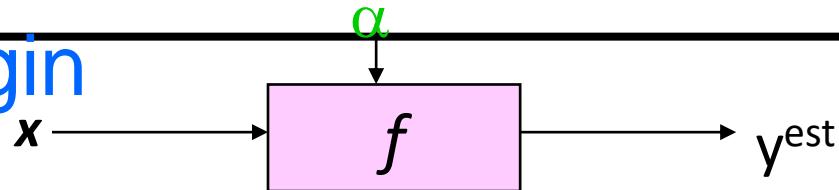


- denotes +1
- denotes -1

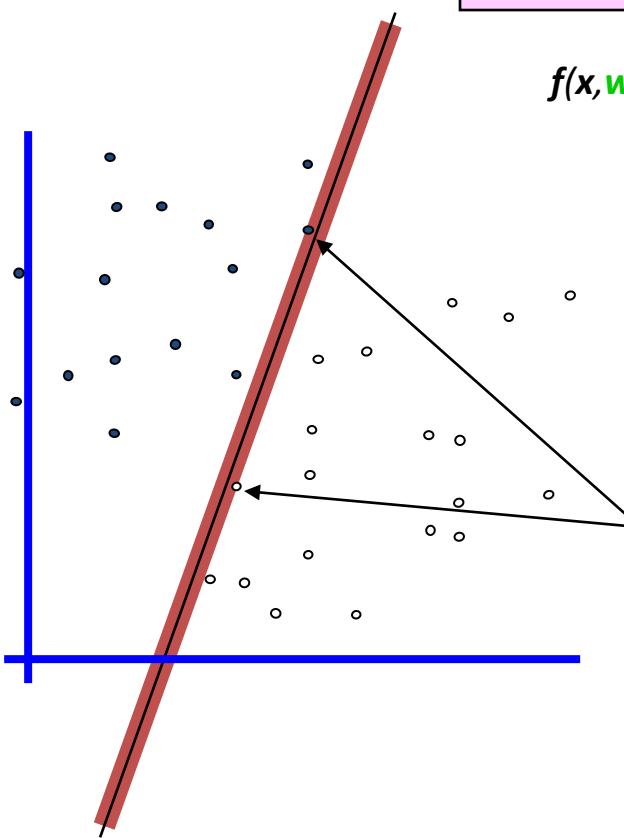




# Classifier Margin



- denotes +1
- denotes -1

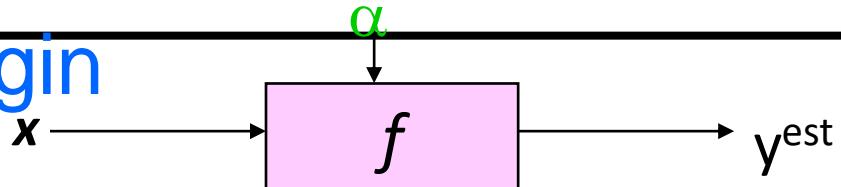


$$f(x, w, b) = \text{sign}(w \cdot x - b)$$

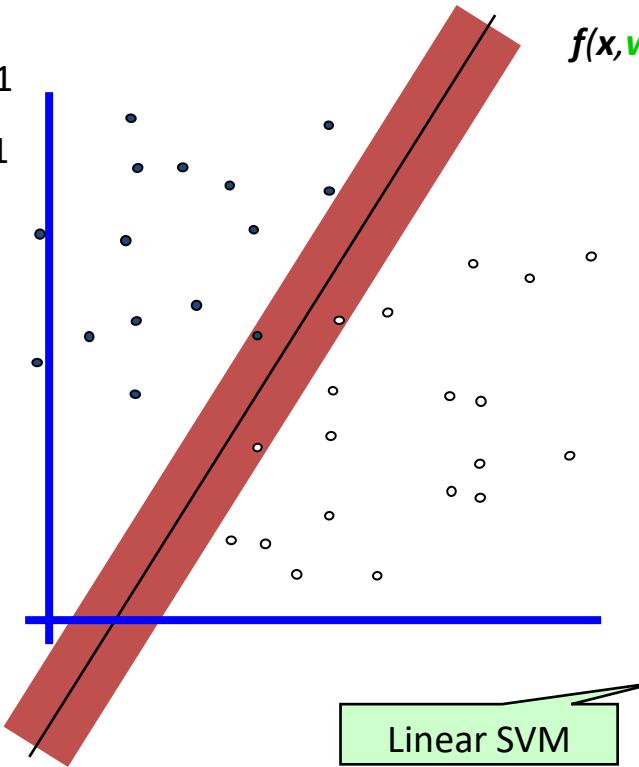
Define the **margin** of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.



# Maximum Margin



- denotes +1
- denotes -1



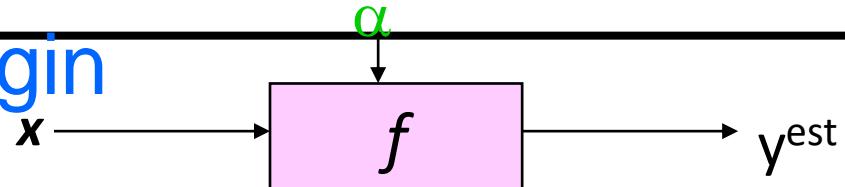
$$f(x, w, b) = \text{sign}(w \cdot x - b)$$

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

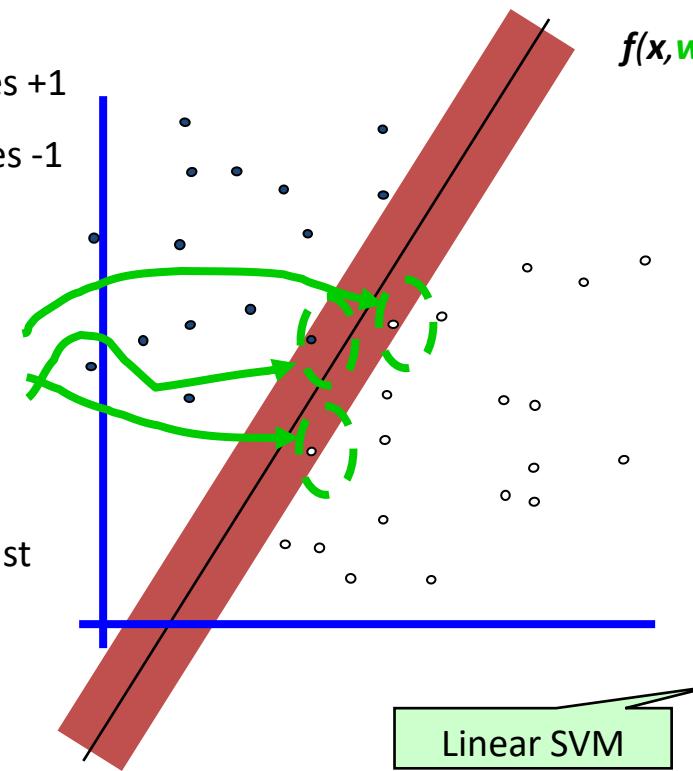
This is the simplest kind of SVM (Called an LSVM)



# Maximum Margin



- denotes +1
  - denotes -1
- Support Vectors are those datapoints that the margin pushes up against



$$f(x, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

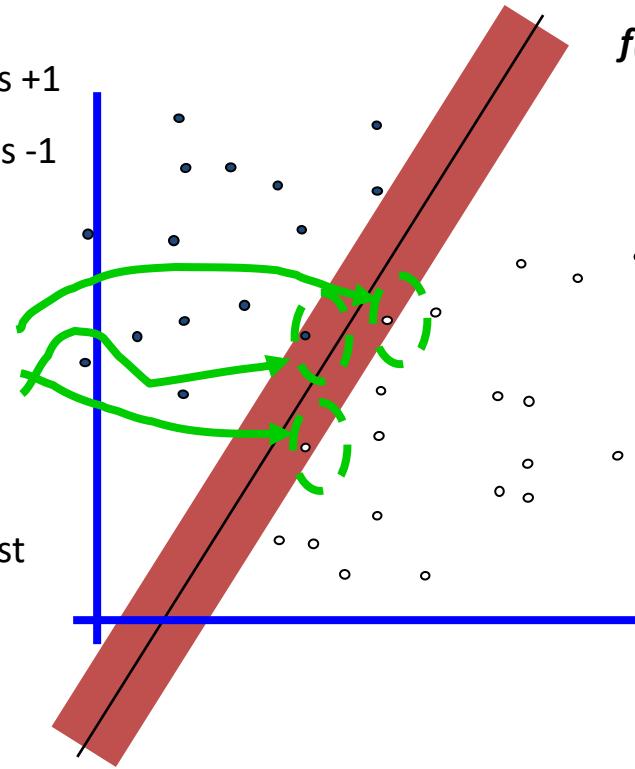
The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

# Why Maximum Margin?



- denotes +1
  - denotes -1
- Support Vectors  
are those  
datapoints that  
the margin  
pushes up against



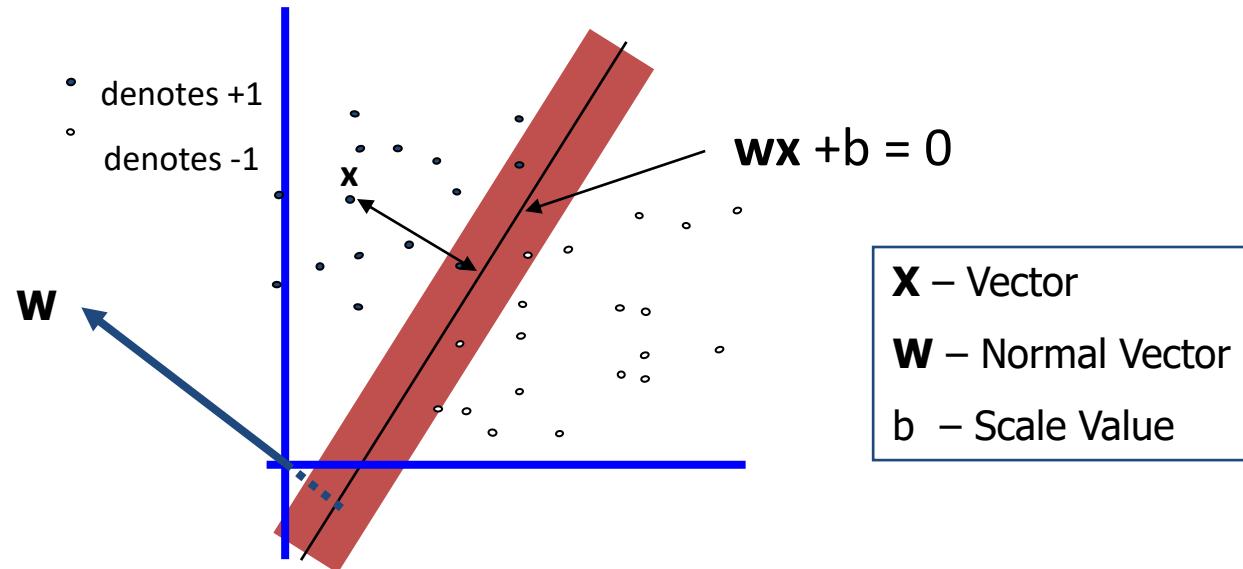
$$f(x, w, b) = \text{sign}(w \cdot x - b)$$

The **maximum margin linear classifier** is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)



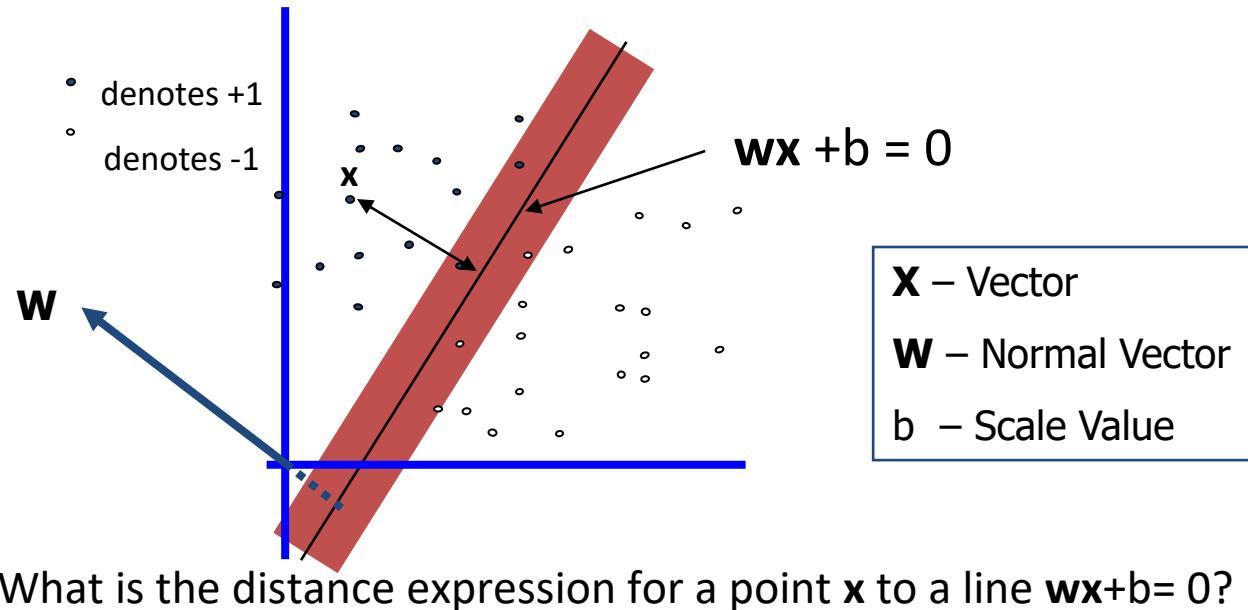
# How to calculate the distance from a point to a line?



- <http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html>
- In our case,  $w_1*x_1+w_2*x_2+b=0$ ,
- thus,  $\mathbf{w}=(w_1, w_2)$ ,  $\mathbf{x}=(x_1, x_2)$



# Estimate the Margin



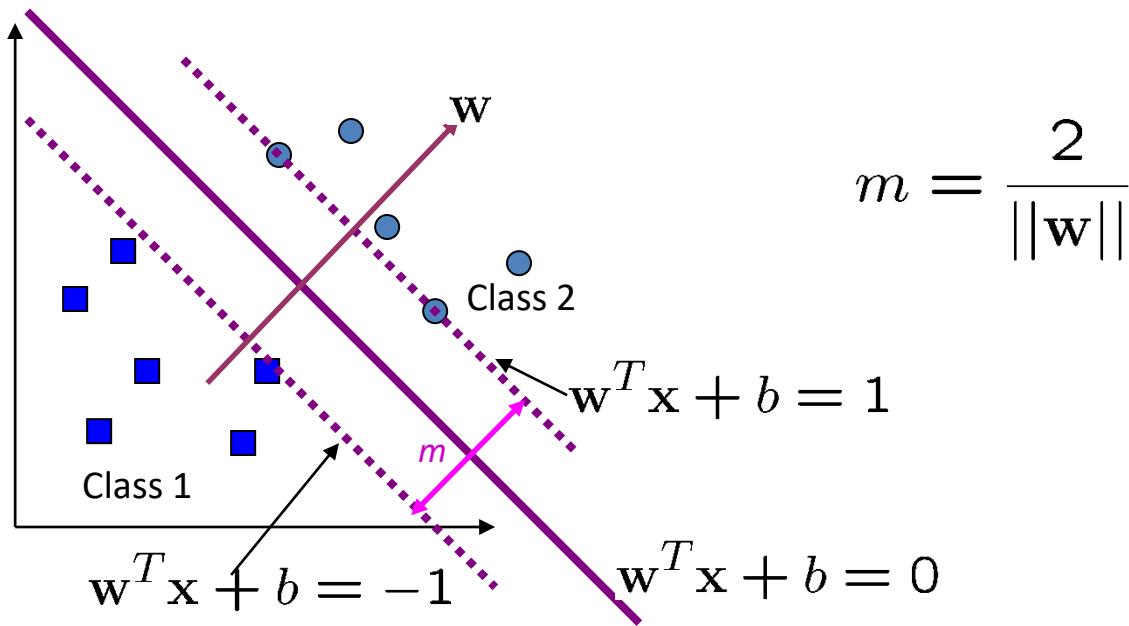
$$d(\mathbf{x}) = \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{\sqrt{\sum_{i=1}^d w_i^2}}$$

# Large-margin Decision Boundary

The decision boundary should be as far away from the data of both classes as possible

We should maximize the margin,  $m$

Distance between the origin and the line  $\mathbf{w}^T \mathbf{x} = -b$  is  $b / \| \mathbf{w} \|$





# Finding the Decision Boundary

---



Let  $\{x_1, \dots, x_n\}$  be our data set and let  $y_i \in \{1, -1\}$  be the class label of  $x_i$

The decision boundary should classify all points correctly  $\Rightarrow$

To see this: when  $y=-1$ , we wish  $(wx+b) < 1$ , when  $y=1$ , we wish  $(wx+b) > 1$ . For support vectors, we wish  $y(wx+b)=1$ .

The decision boundary can be found by solving the following constrained optimization problem

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad \forall i$$

$$\text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad \forall i$$



# Next step... Optional

---

Converting SVM to a form we can solve



Dual form

Allowing a few errors

Soft margin

Allowing nonlinear boundary

Kernel functions



## The Dual Problem (we ignore the derivation)



The new objective function is in terms of  $\alpha_i$  only

It is known as the dual problem: if we know  $\mathbf{w}$ , we know all  $\alpha_i$ ; if we know all  $\alpha_i$ , we know  $\mathbf{w}$

The original problem is known as the primal problem

The objective function of the dual problem needs to be maximized!

The dual problem is therefore:

$$\max. W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to  $\alpha_i \geq 0,$

Properties of  $\alpha_i$  when we introduce the Lagrange multipliers

$$\sum_{i=1}^n \alpha_i y_i = 0$$

The result when we differentiate the original Lagrangian w.r.t. b



# The Dual Problem

---



$$\begin{aligned} \text{max. } W(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to } \alpha_i &\geq 0, \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

This is a quadratic programming (QP) problem

A global maximum of  $\alpha_i$  can always be found

$\mathbf{w}$  can be recovered by

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$



# Characteristics of the Solution

---



Many of the  $\alpha_i$  are zero (see next page for example)

$\mathbf{w}$  is a linear combination of a small number of data points

This “sparse” representation can be viewed as data compression as in the construction of knn classifier

$\mathbf{x}_i$  with non-zero  $\alpha_i$  are called support vectors (SV)

The decision boundary is determined only by the SV

Let  $t_j$  ( $j=1, \dots, s$ ) be the indices of the  $s$  support vectors. We can write

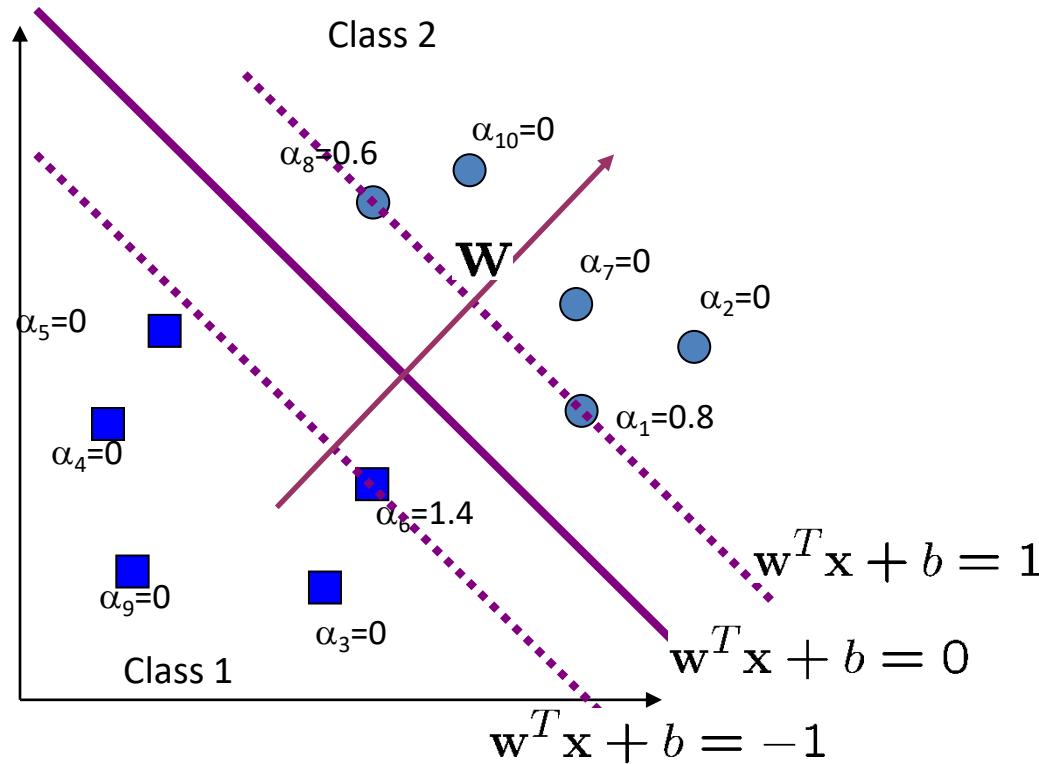
For testing with a new data  $\mathbf{z}$   $\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$

Compute and classify

$\mathbf{z}$  as class  $\mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$ , otherwise

Note:  $\mathbf{w}$  need not be formed explicitly

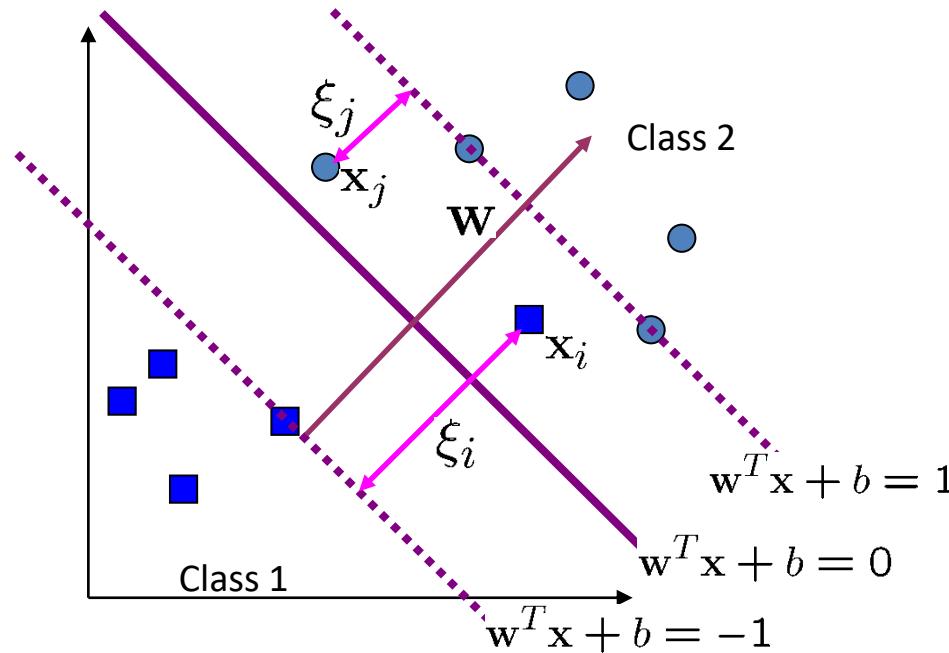
# A Geometrical Interpretation



# Allowing errors in our solutions

We allow “error”  $\xi_i$  in classification; it is based on the output of the discriminant function  $\mathbf{w}^T \mathbf{x} + b$

$\xi_i$  approximates the number of misclassified samples





# Soft Margin Hyperplane

---

If we minimize  $\sum_i \xi_i$ ,  $\xi_i$  can be computed by



$\xi_i$  are “slack variables” if  
Note that  $\xi_i=0$  if there is  
 $\xi_i$  is an upper bound of the number of errors

$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T \mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

We want to minimize

$C$  : tradeoff parameter between error and margin

The optimization problem becomes

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ & \text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \end{aligned}$$



# Extension to Non-linear Decision Boundary

---



So far, we have only considered large-margin classifier with a linear decision boundary

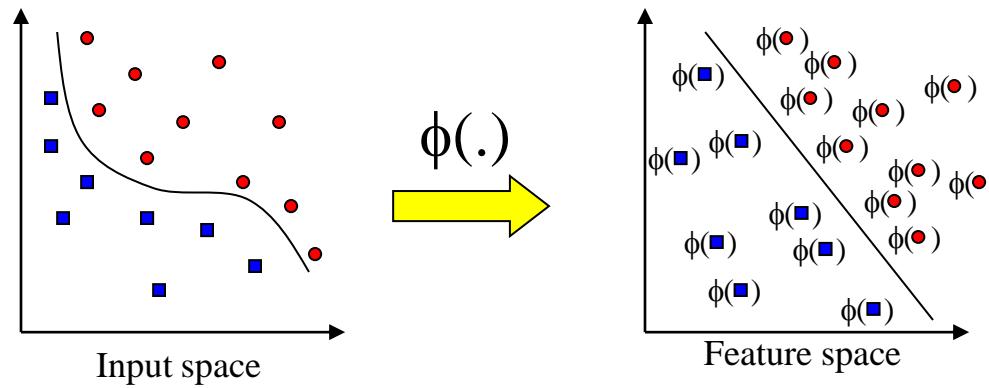
How to generalize it to become nonlinear?

Key idea: transform  $\mathbf{x}_i$  to a higher dimensional space to “make life easier”

Input space: the space the point  $\mathbf{x}_i$  are located

Feature space: the space of  $\phi(\mathbf{x}_i)$  after transformation

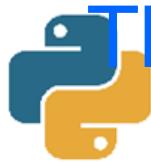
# Transforming the Data (c.f. DHS Ch. 5)



Note: feature space is of higher dimension than the input space in practice

Computation in the feature space can be costly because it is high dimensional

The feature space is typically infinite-dimensional!  
The kernel trick comes to rescue



# The Kernel Trick

---

Recall the SVM optimization problem



$$\max. \quad W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

The data points only appear

As long as we can calculate inner products subject to  $C \geq \alpha_i \geq 0$ ,  $\sum_{i=1}^n \alpha_i y_i = 0$  we do not need the mapping explicitly

Many common geometric operations (angles, distances) can be expressed by inner products

Define the kernel function  $K$  by

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$



# An Example for $\phi(\cdot)$ and $K(\cdot, \cdot)$

---

Suppose  $\phi(\cdot)$  is given as follows



An inner product in the space  $\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$

So, if we define the kernel function as follows there is no need to carry out  $\phi(\cdot)$  explicitly  
 $\langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) \rangle = (1 + x_1y_1 + x_2y_2)^2$

This use of kernel function to avoid carrying out  $\phi(\cdot)$  explicitly is known as the [kernel trick](#)

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$



# More on Kernel Functions

---

Not all similarity measures can be used as kernel function, however

The kernel function needs to satisfy the [Mercer function](#), i.e., the function is “positive-definite”

This implies that

the  $n$  by  $n$  kernel matrix,

in which the  $(i,j)$ -th entry is the  $K(\mathbf{x}_i, \mathbf{x}_j)$ , is always positive definite

This also means that optimization problem can be solved in polynomial time!





# Examples of Kernel Functions

---



Polynomial kernel with degree  $d$

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

Radial basis function kernel with width  $\sigma$

$$K(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / (2\sigma^2))$$

Closely related to radial basis function neural networks

The feature space is infinite-dimensional

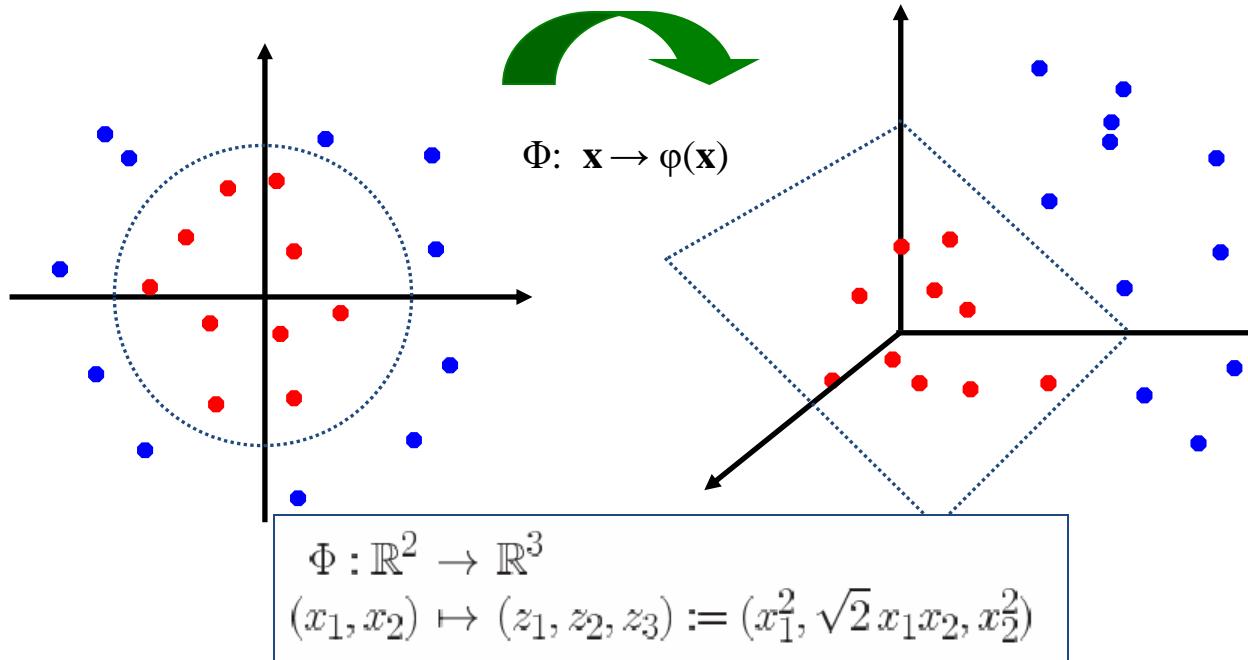
Sigmoid with parameter  $\kappa$  and  $\theta$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} + \theta)$$

It does not satisfy the Mercer condition on all  $\kappa$  and  $\theta$

## Non-linear SVMs: Feature spaces

- **General idea:** the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



# Example

Suppose we have 5 one-dimensional data points

$x_1=1, x_2=2, x_3=4, x_4=5, x_5=6$ , with 1, 2, 6 as class 1 and 4, 5 as class 2  $\Rightarrow y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1$

We use the polynomial kernel of degree 2

$$K(x,y) = (xy+1)^2$$

C is set to 100

We first find  $\alpha_i$  ( $i=1, \dots, 5$ ) by

$$\max. \sum_{i=1}^5 \alpha_i - \frac{1}{2} \sum_{i=1}^5 \sum_{j=1}^5 \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^5 \alpha_i y_i = 0$$

# Example



By using a QP solver, we get

$$\alpha_1=0, \alpha_2=2.5, \alpha_3=0, \alpha_4=7.333, \alpha_5=4.833$$

Note that the constraints are indeed satisfied

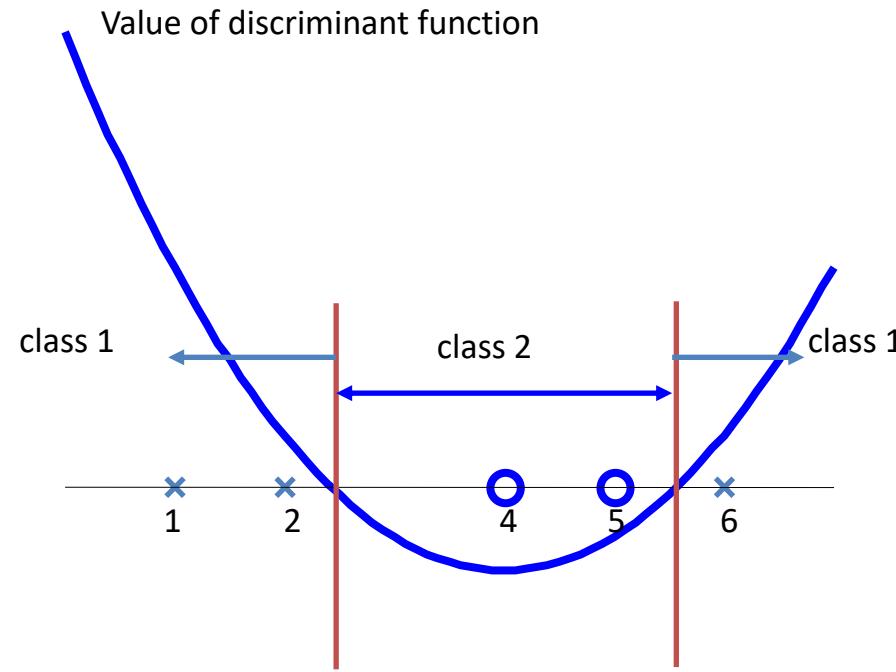
The support vectors are  $\{x_2=2, x_4=5, x_5=6\}$

The discriminant function is

$b$  is recovered by solving  $f(z) = 1$  or by  $f(5) = -1$  or by  $f(6) = 1$ , as  $x_2$  and  $x_5$  lie on the line and  $x_4$  lies on the line  
$$f(z) = 2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + 4.833(1)(6z+1)^2 + b$$
$$= 0.6667z^2 - 5.333z + b$$
  
All three give  $b=9$

$$\begin{aligned}\phi(\mathbf{w})^T \phi(\mathbf{x}) + b &= 1 \\ \phi(\mathbf{w})^T \phi(\mathbf{x}) + b &= -1 \\ \rightarrow f(z) &= 0.6667z^2 - 5.333z + 9\end{aligned}$$

# Example





# Choosing the Kernel Function

---

Probably the most tricky part of using SVM.





# Summary: Steps for Classification

---



Prepare the pattern matrix

Select the kernel function to use

Select the parameter of the kernel function and the value of C

You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter

Execute the training algorithm and obtain the  $\alpha_i$

Unseen data can be classified using the  $\alpha_i$  and the support vectors



# Conclusion

---

SVM is a useful alternative to neural networks

Two key concepts of SVM: maximize the margin and the kernel trick

Many SVM implementations are available on the web for you to try on your data set!





# k-Nearest Neighbors

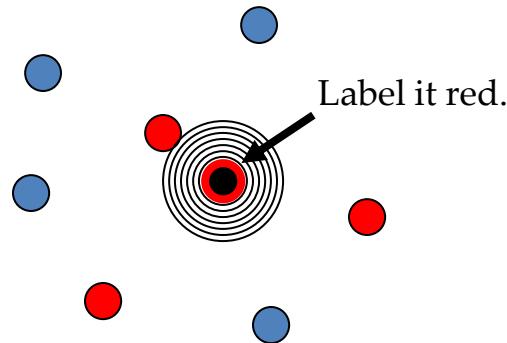
---

MODELOS BÁSICOS

# 1-Nearest Neighbor

One of the simplest of all machine learning classifiers

Simple idea: label a new point the **same as the closest known point**

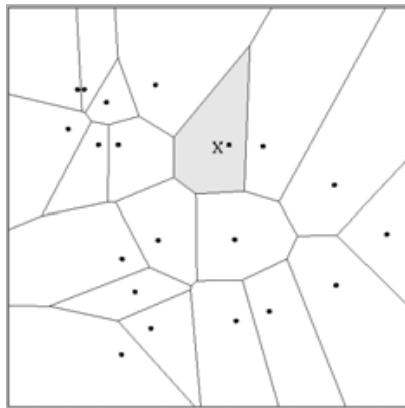




# Distance Metrics



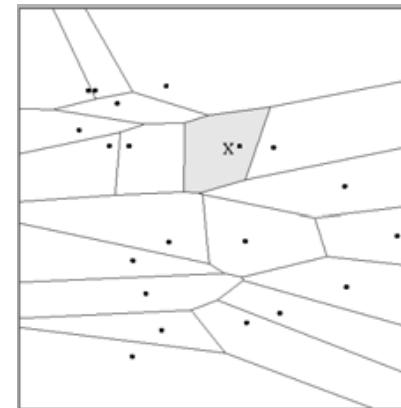
Different metrics can change the decision surface



Standard Euclidean distance metric:  $\text{Dist}(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$

Two-dimensional:  $\text{Dist}(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$

Multivariate:  $\text{Dist}(a,b) = \sqrt{\sum (a_i - b_i)^2}$



$\text{Dist}(a,b) = (a_1 - b_1)^2 + (3a_2 - 3b_2)^2$

Adapted from "Instance-Based Learning"  
lecture slides by Andrew Moore, CMU.



## 1-NN's Aspects as an Instance-Based Learner:



A distance metric

Euclidean

When different units are used for each dimension

→ normalize each dimension by standard deviation

For discrete data, can use hamming distance

→  $D(x_1, x_2) = \text{number of features on which } x_1 \text{ and } x_2 \text{ differ}$

Others (e.g., normal, cosine)

How many nearby neighbors to look at?

One

How to fit with the local points?

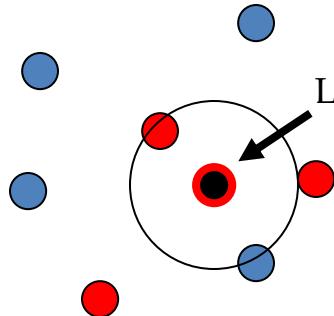
Just predict the same output as the nearest neighbor.

Adapted from "Instance-Based Learning"  
lecture slides by Andrew Moore, CMU.

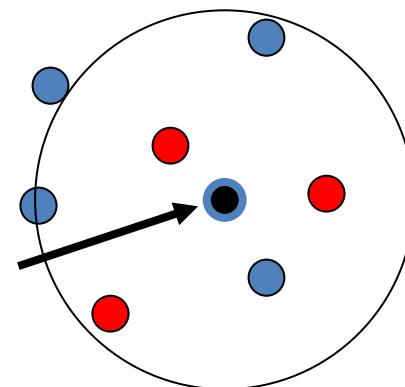
# k – Nearest Neighbor

Generalizes 1-NN to smooth away noise in the labels

A new point is now assigned **the most frequent label of its  $k$  nearest neighbors**



Label it blue, when  $k = 7$





# KNN Example



	Food (3)	Chat (2)	Fast (2)	Price (3)	Bar (2)	BigTip
1	great	yes	yes	normal	no	yes
2	great	no	yes	normal	no	yes
3	mediocre	yes	no	high	no	no
4	great	yes	yes	normal	yes	yes

Similarity metric: Number of matching attributes (k=2)

New examples:

Example 1 (great, no, no, normal, no)      Yes

→ most similar: number 2 (1 mismatch, 4 match) → yes

→ Second most similar example: number 1 (2 mismatch, 3 match) → yes

Example 2 (mediocre, yes, no, normal, no)

Yes/No

→ Most similar: number 3 (1 mismatch, 4 match) → no

→ Second most similar example: number 1 (2 mismatch, 3 match) → yes

# Selecting the Number of Neighbors



Increase k:

Makes KNN less sensitive to noise

Decrease k:

Allows capturing finer structure of space

→ Pick k not too large, but not too small (depends on data)



# Curse-of-Dimensionality

---

Prediction accuracy can quickly degrade when number of attributes grows.



Irrelevant attributes easily “swamp” information from relevant attributes

When many irrelevant attributes, similarity/distance measure becomes less reliable

## Remedy

Try to remove irrelevant attributes in pre-processing step

Weight attributes differently

Increase k (but not too much)





# Advantages and Disadvantages of KNN

---

Need distance/similarity measure and attributes that “match” target function.



For large training sets,

→ Must make a pass through the entire dataset for each classification. This can be prohibitive for large data sets.

Prediction accuracy can quickly degrade when number of attributes grows.

Simple to implement algorithm;

Requires little tuning;

Often performs quite well!

(Try it first on a new learning problem).



# Convolutional Neural Network (CNN)

---

MODELOS BÁSICOS



---

## Feed-Forward Neural Network shortcomings

## Convolutional Neural Networks

How does a CNN work

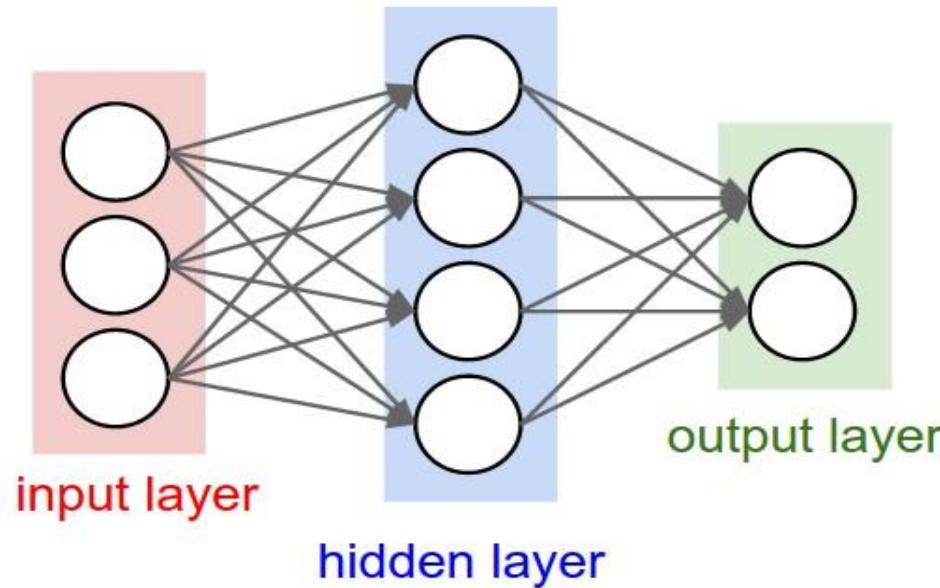
Convolution layer

Pooling layer

## Implementing CNN using TensorFlow

## Save and Restore the Network

# Feed Forward Neural Network (NN)



# Neural Network Problems:



1. Doesn't use data structure!



# Neural Network Problems:

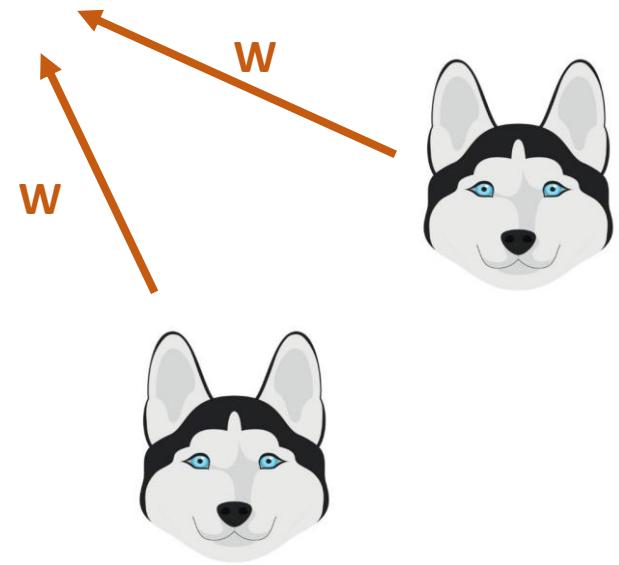


- 
1. Doesn't use data structure!

**Solution:** Weight sharing

**CNN:**

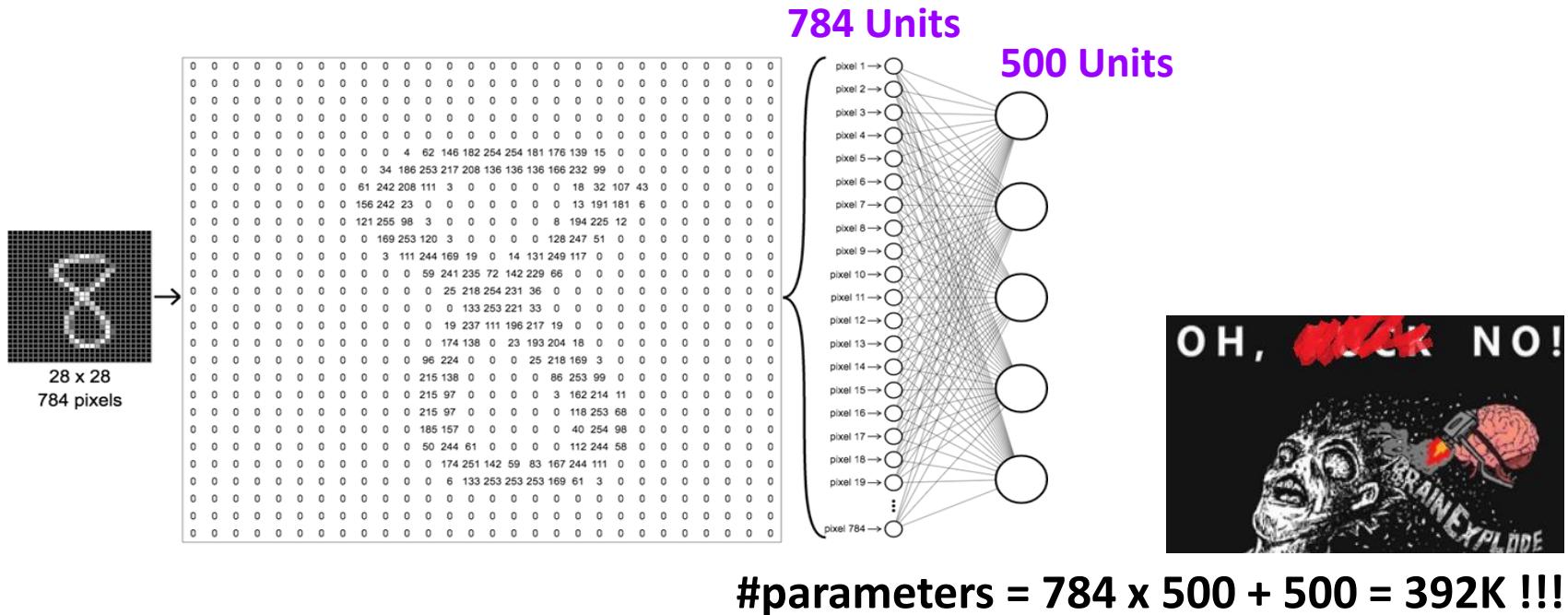
Neural Net that shares it's parameters across space



# Neural Network Problems:



2. Doesn't scale well to full images

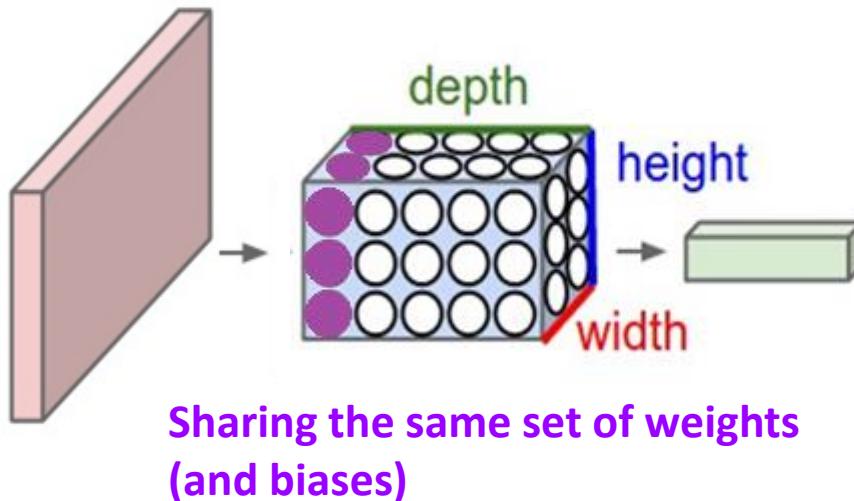


# Neural Network Problems:

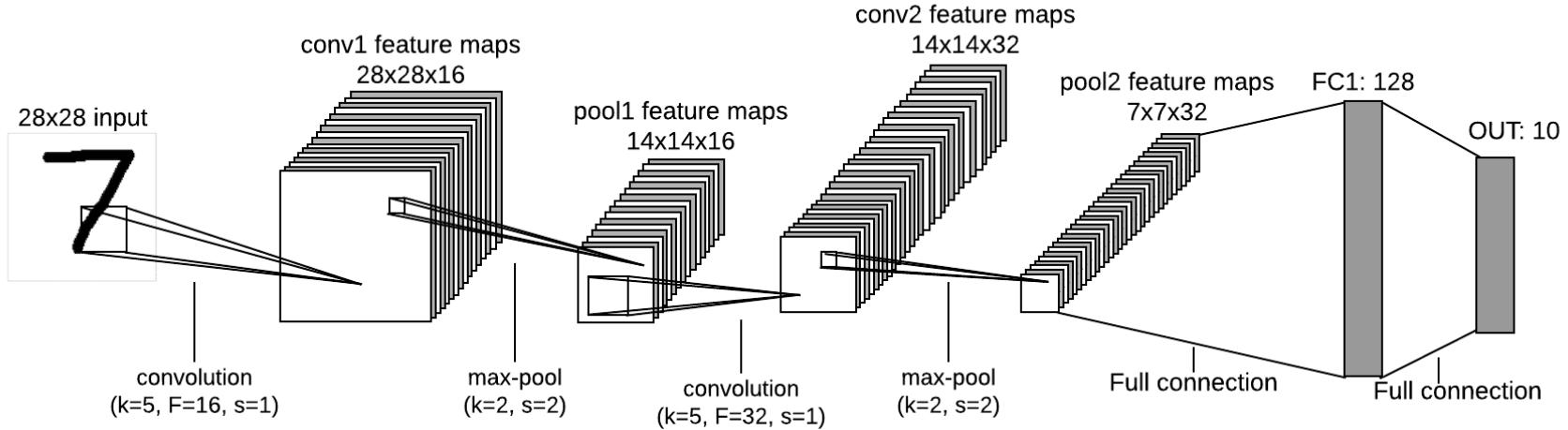


2. Doesn't scale well to full images

**Solution:** Weight sharing + 3D volume of neurons



# Layers Used to Build CNN



# Convolution Layer

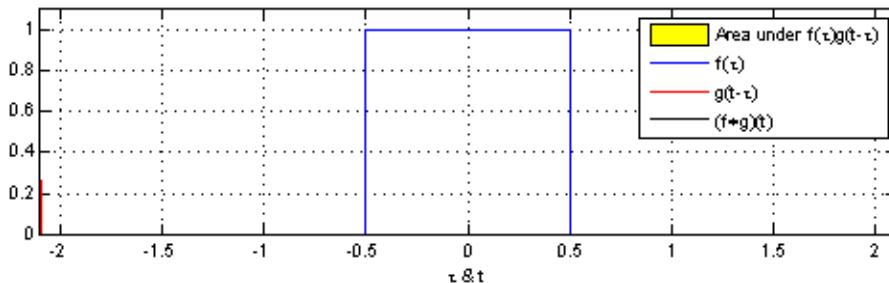


What is convolution?

A function derived from two given functions by integration that expresses how the shape of one is modified by the other.

Continues:  $(f * g)(t) = \int f(\tau)g(t - \tau) d\tau$

$$\text{Discrete: } (f * g)[n] = \sum f[m] g[n - m]$$

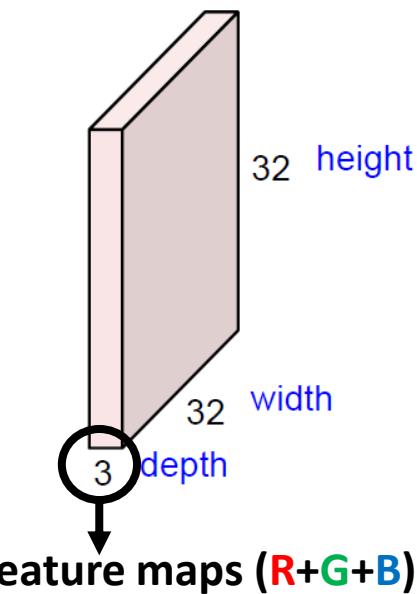


1. slide
2. multiply
3. integrate (i.e. sum)

# Convolution Layer



32x32x3 image

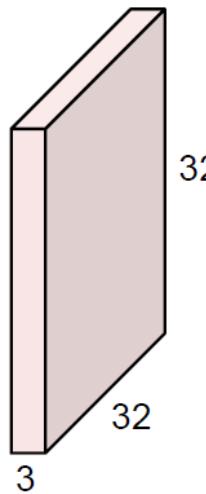


- Spatial dimensions: 32x32
- Depth: 3

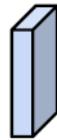
# Convolution Layer



32x32x3 image



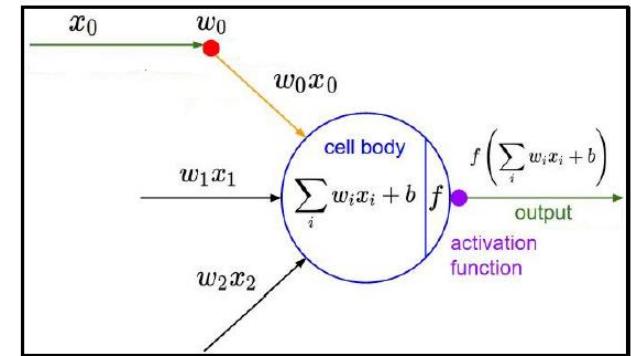
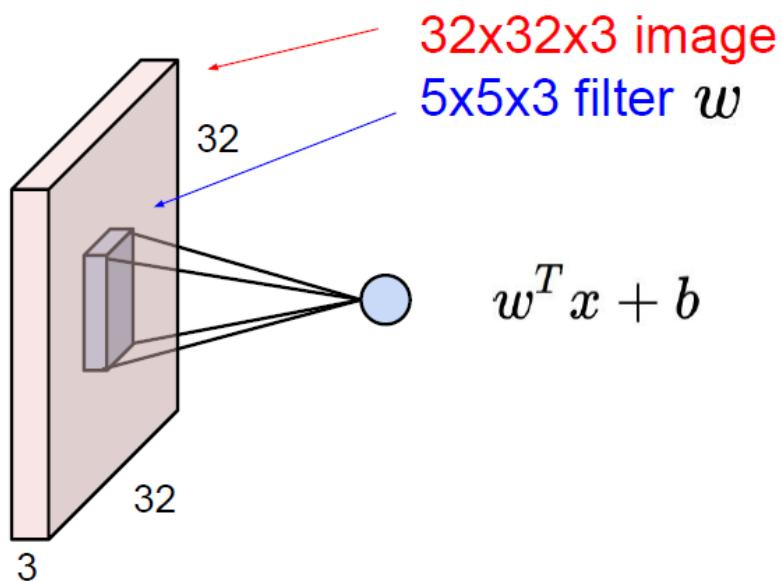
5x5x3 filter



**(Filter = Kernel = Patch)**

**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
Computing dot products,  
And sum over all”

# Convolution Layer



# Convolution Layer



3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

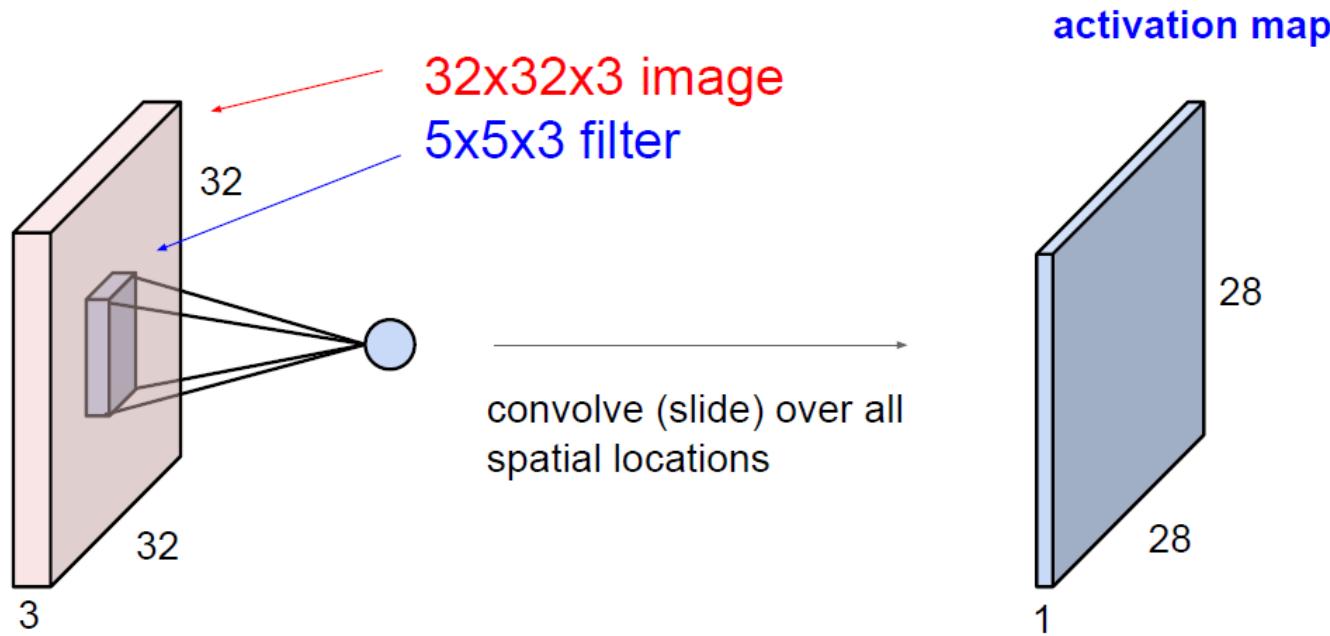
3	3 <sub>0</sub>	2 <sub>1</sub>	1 <sub>2</sub>	0
0	0 <sub>2</sub>	1 <sub>2</sub>	3 <sub>0</sub>	1
3	1 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

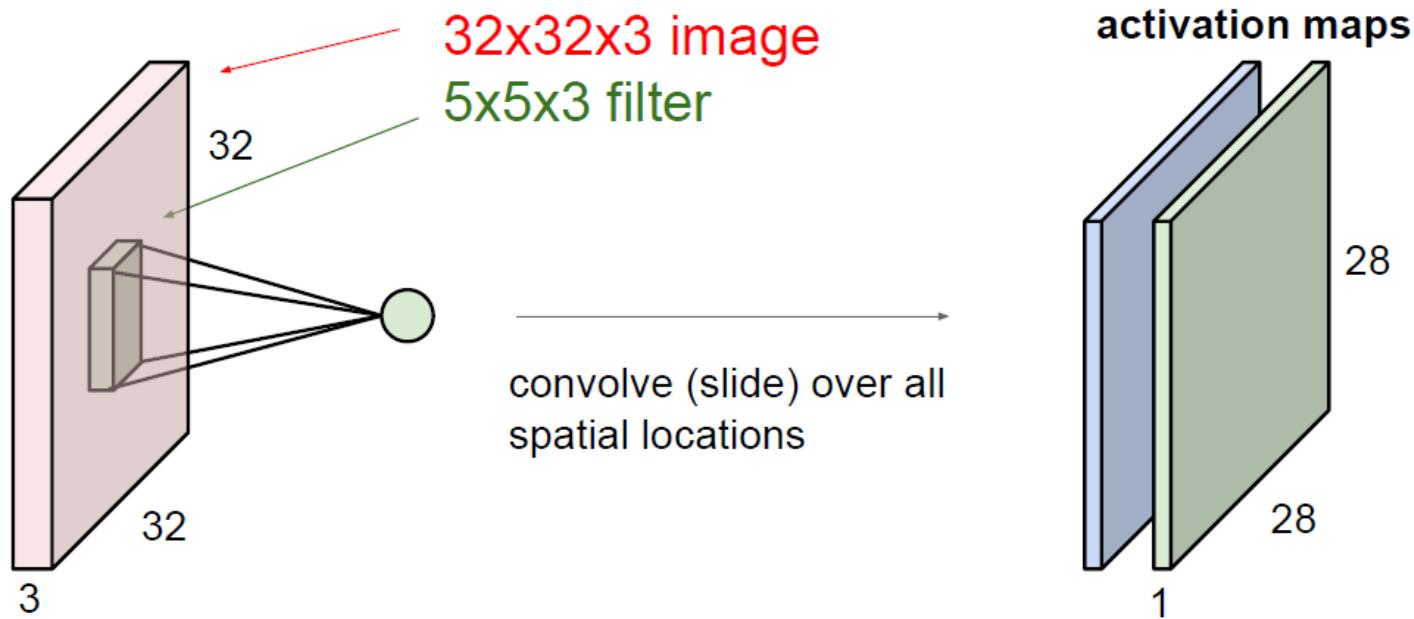
3	3	2 <sub>0</sub>	1 <sub>1</sub>	0 <sub>2</sub>
0	0	1 <sub>2</sub>	3 <sub>2</sub>	1 <sub>0</sub>
3	1	2 <sub>0</sub>	2 <sub>1</sub>	3 <sub>2</sub>
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

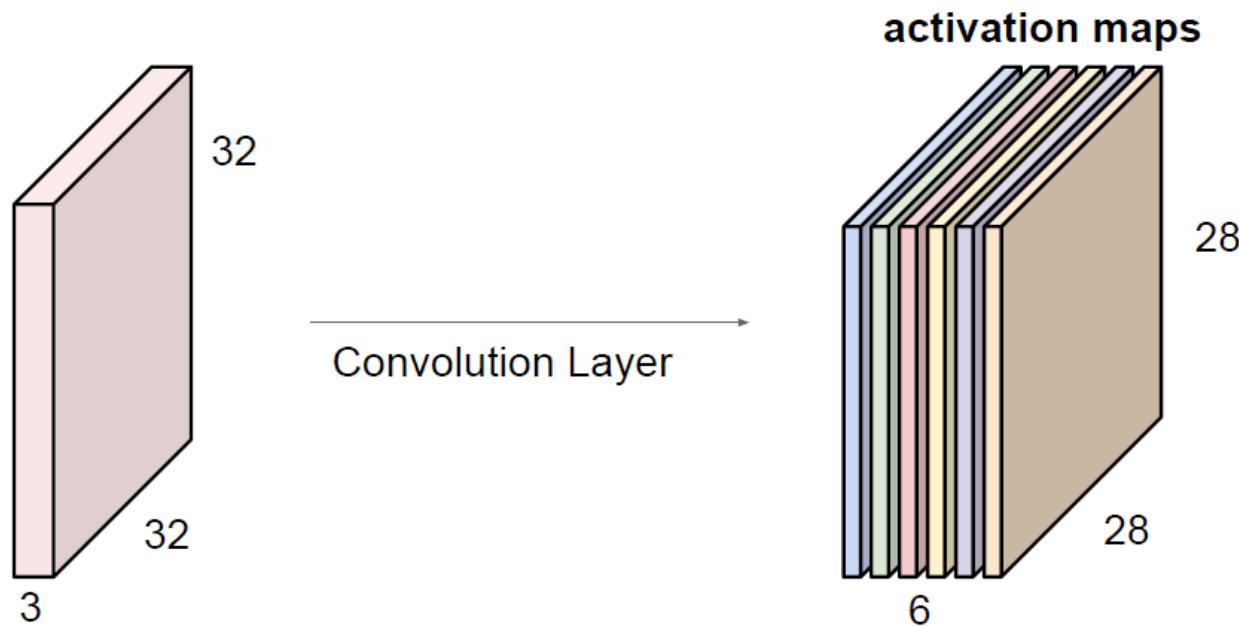
# Convolution Layer



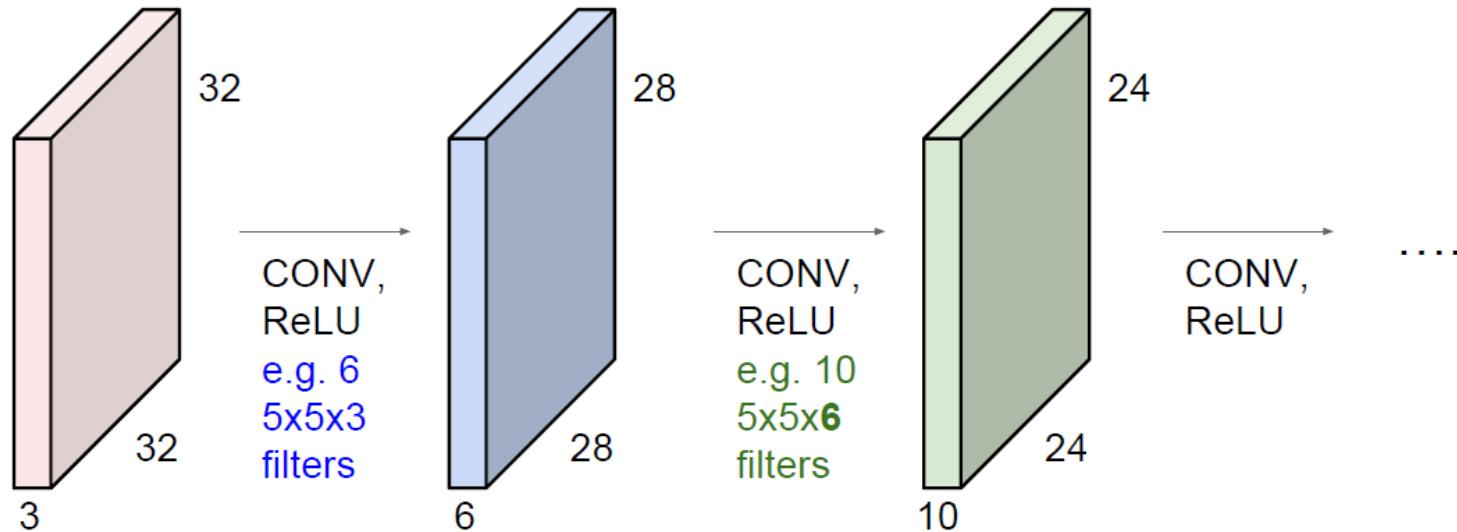
# Convolution Layer



# Convolution Layer

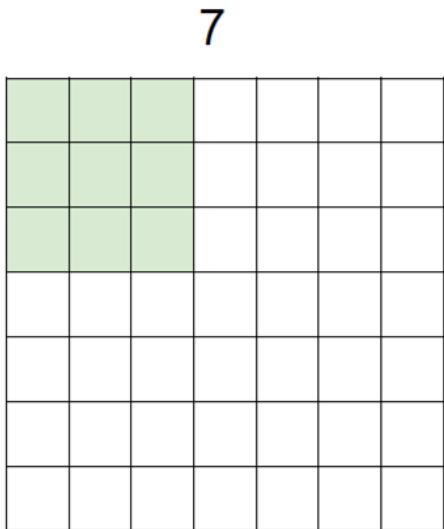


# Convolution Layer



# Convolution Layer

A closer look at spatial dimensions:



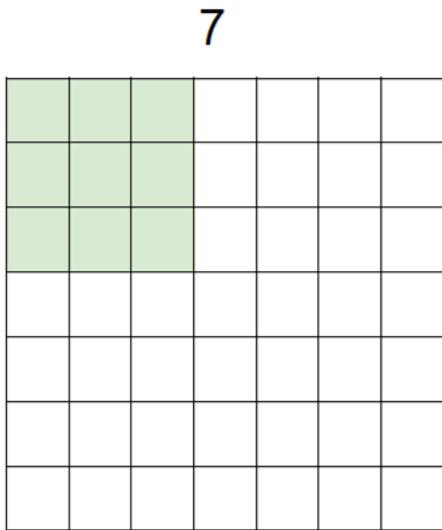
- Input size:  $i = 7$
- Filter size:  $K=3$
- Stride:  $s=2$

=> Output size:  $o = 3$

$$o = (i - K) / s + 1$$

# Convolution Layer

A closer look at spatial dimensions:



- Input size:  $i = 7$
- Filter size:  $K=3$
- Stride:  $s=3$

$$o=(i-K)/s +1 = 2.33 \text{ !!!}$$

# Convolution Layer

A closer look at spatial dimensions:

0	0	0	0	0	0			
0								
0								
0								
0								

- Zero-padding:
    - Input size:  $i = 7$
    - Filter size:  $K=3$
    - Stride:  $s=3$
    - Zero-pad:  $p=1$
- => Output size:  $o = 3$

$$o = (i + 2p - K) / s + 1$$



# Convolution Layer

---

A closer look at spatial dimensions:



**Valid padding:** don't go past the edges

**Same padding:** go off the edge (with  $s=1$ ), pad with zeros in such a way that **output size = input size**

$$o = (i + 2p - K)/s + 1 \Rightarrow i = (o + K - 1)s - 1 \Rightarrow p = (K-1)/2$$

$K=3 \Rightarrow$  zero pad with 1

$K=5 \Rightarrow$  zero pad with 2

$K=7 \Rightarrow$  zero



# Convolution in TensorFlow

---



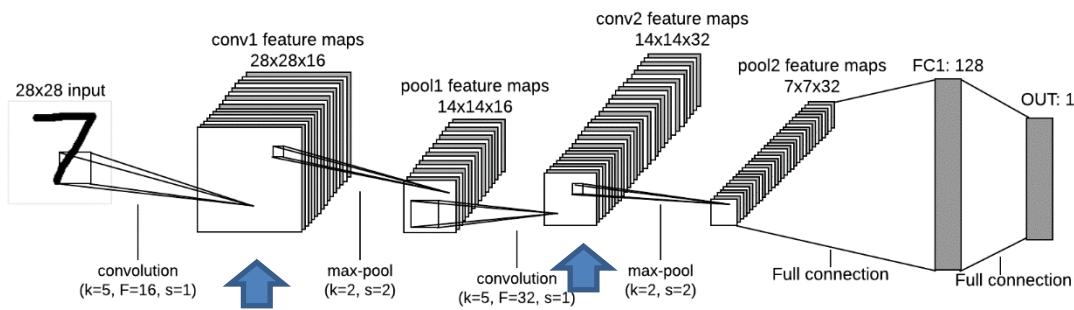
Computes a 2-D convolution given 4-D input and filter tensors

**input:** tensor of shape  
[batch\_size, in\_height, in\_width, in\_channels]

**filter:** tensor of shape  
[filter\_height, filter\_width, in\_channels, out\_channels]

```
tf.nn.conv2d(  
    input,  
    filter,  
    strides,  
    padding,  
    use_cudnn_on_  
gpu=True,  
    data_format='  
NHWC',  
    dilations=[1,  
1, 1, 1],  
    name=None  
)
```

# Helper Function for Convolution Layer



```
def conv_layer(x, filter_size, num_filters, stride, name):

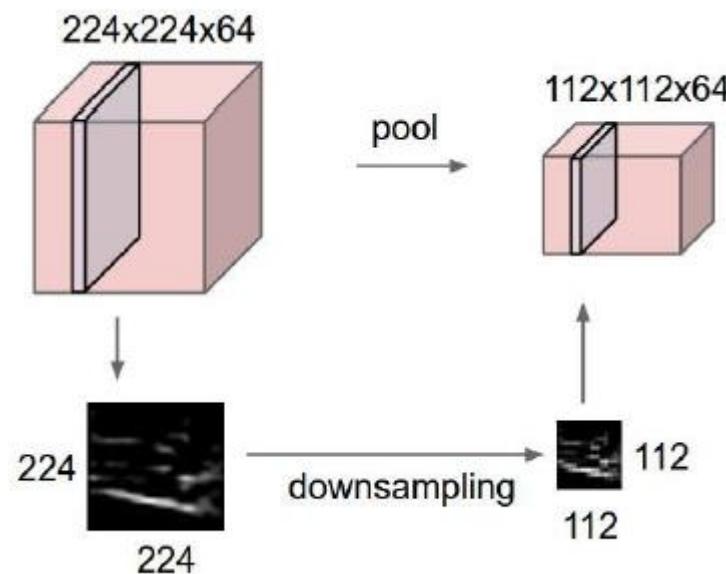
    with tf.variable_scope(name):
        num_in_channel = x.get_shape().as_list()[-1]
        shape = [filter_size, filter_size, num_in_channel, num_filters]
        W = weight_variable(shape=shape)
        tf.summary.histogram('weight', W)
        b = bias_variable(shape=[num_filters])
        tf.summary.histogram('bias', b)
        layer = tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding="SAME")
        layer += b
    return tf.nn.relu(layer)
```

# Pooling Layer

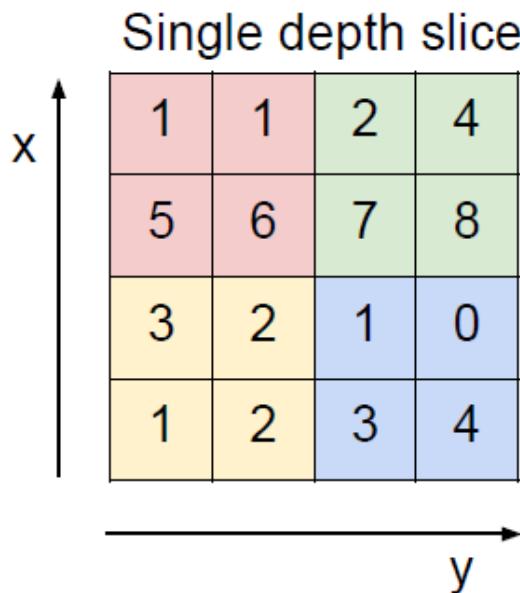


To reduce the spatial dimension of feature maps  
of parameters

reduce the amount



# Max Pooling



max pool with 2x2 filters  
and stride 2

6	8
3	4

F = Filter Size  
S = Stride

Common Settings:  
F = 2, S = 2  
F = 3, S = 2

# Max Pooling



3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

$$o = (i - F)/s + 1$$

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Input size: **i = 5**

Filter size: **F=3**

Stride: **s=1**

=> Output size: **o = 3**

# Average Pooling



3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

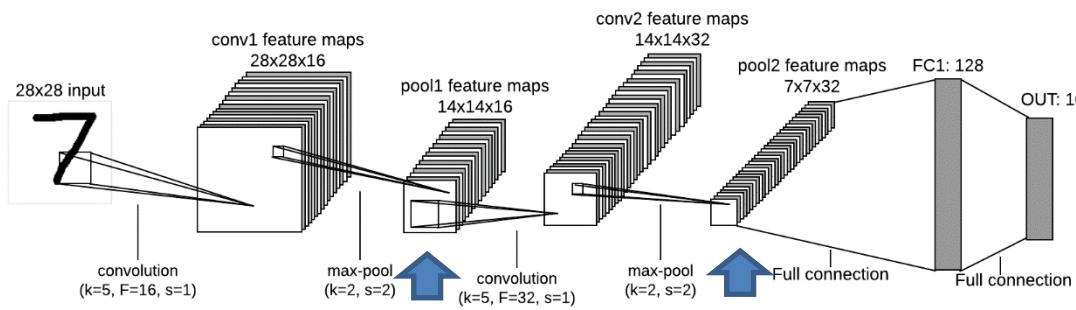
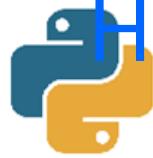
3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

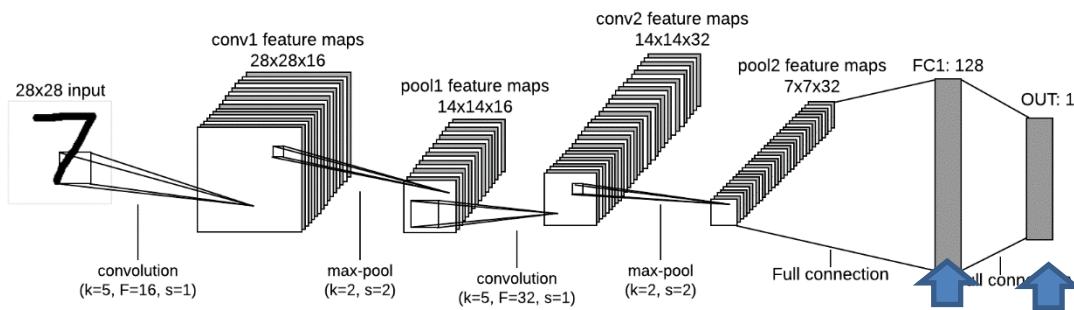
# Helper Function for Pooling Layer



```
def max_pool(x, ksize, stride, name):

    return tf.nn.max_pool(x,
                          ksize=[1, ksize, ksize, 1],
                          strides=[1, stride, stride, 1],
                          padding="SAME",
                          name=name)
```

# Helper Function for Flatten Layer and Fully Connected Layer



```
def flatten_layer(layer):

    with tf.variable_scope('Flatten_layer'):
        layer_shape = layer.get_shape()
        num_features = layer_shape[1:4].num_elements()
        layer_flat = tf.reshape(layer, [-1, num_features])
    return layer_flat
```

```
def fc_layer(x, num_units, name, use_relu=True):

    with tf.variable_scope(name):
        in_dim = x.get_shape()[1]
        W = weight_variable(shape=[in_dim, num_units])
        tf.summary.histogram('weight', W)
        b = bias_variable(shape=[num_units])
        tf.summary.histogram('bias', b)
        layer = tf.matmul(x, W)
        layer += b
        if use_relu:
            layer = tf.nn.relu(layer)
    return layer
```



# Remarks on Implementation

---



```
# hyper-parameters

learning_rate = 0.001 # The optimization learning rate
epochs = 10          # Total number of training epochs
batch_size = 100      # Training batch size

# Network Parameters
# We know that MNIST images are 28 pixels in each dimension.
img_h = img_w = 28

# Images are stored in one-dimensional arrays of this length.
img_size扁 = img_h * img_w

# Number of classes, one class for each of 10 digits.
n_classes = 10
```



# Remarks on Implementation

---



## Feed-forward network:

```
# Placeholders for inputs (x), outputs (y)
with tf.name_scope('input'):
    x = tf.placeholder(tf.float32, shape=[None, img_size_flat], name='X')
    y = tf.placeholder(tf.float32, shape=[None, n_classes], name='Y')
```

## CNN

```
# Placeholders for inputs (x), outputs (y)
with tf.name_scope('input'):
    x = tf.placeholder(tf.float32, shape=[None, img_h, img_w, n_channels], name='X')
    y = tf.placeholder(tf.float32, shape=[None, n_classes], name='Y')
```

# Save and Restore Model



Process of training a network is time consuming

TensorFlow has an easy solution to save and restore your model:

**In the graph:**

```
saver = tf.train.Saver()
```

```
↳ saver = {Saver} <tensorflow.python.training.saver.Saver object at 0x00000000058C9518>
  ↳ _allow_empty = {bool} False
  ↳ _builder = {BaseSaverBuilder} <tensorflow.python.training.saver.BaseSaverBuilder object at 0x000000001EAAF1D0>
  ↳ _checkpoints_to_be_deleted = {list} <class 'list'>: []
  ↳ _filename = {NoneType} None
  ↳ _is_built = {bool} True
  ↳ _is_empty = {bool} False
  ↳ _keep_checkpoint_every_n_hours = {float} 10000.0
  ↳ _last_checkpoints = {list} <class 'list'>: []
  ↳ _max_to_keep = {int} 5
  ↳ _name = {NoneType} None
  ↳ _next_checkpoint_time = {float} 1557762683.4663734
  ↳ _pad_step_number = {bool} False
  ↳ _reshape = {bool} False
  ↳ _restore_sequentially = {bool} False
  ↳ _save_relative_paths = {bool} False
  ↳ _sharded = {bool} False
  ↳ _var_list = {list} <class 'list'>: [<tf.Variable 'conv1/W:0' shape=(5, 5, 1, 16) dtype=float32_ref>, <tf.Variable 'conv1/b:0' shape=(16,) dtype=float32_ref>, <tf.Variable 'conv2/W:0' shape=(1, 1, 16, 32) dtype=float32_ref>, <tf.Variable 'conv2/b:0' shape=(32,) dtype=float32_ref>, <tf.Variable 'fc1/W:0' shape=(32, 10) dtype=float32_ref>, <tf.Variable 'fc1/b:0' shape=(10,) dtype=float32_ref>]
  ↳ _write_version = {int} 2
  ↳ last_checkpoints = {list} <class 'list'>: []
  ↳ saver_def = {SaverDef} filename_tensor_name: "save/Const:0"\n  save_tensor_name: "save/control_dependency:0"\n  restore_op_name: "save/restore_all"\n  max_to_keep: 5\n  
```

# Save and Restore Model



In the session:

## Save

In the session: (usually after training is done)

```
saver.save(sess, model_path + model_name, global_step)
```

Model is loaded with the TensorFlow names !

```
var_python_name = tf.get_variable("var_tf_name", shape=...)
```

## Restore

in the session: (usually before running the test functions)

```
saver.restore(sess, tf.train.lavariableis saved with this name in the file
```



# Recurrent Neural Network Architectures

---



---

# Lecture Outline

1. Introduction
  2. Learning Long Term Dependencies
  3. Regularization
  4. Visualization for RNNs
-



---

## Section 1: Introduction



---

# Applications of RNNs



Image Captioning

[\[reference\]](#)

... and more!

VIOLA:

Why, Salisbury must find his flesh and thought  
That which I am not aps, not a man and in fire,  
To show the reining of the raven and the wars  
To grace my hand reproach within, and not a fair are hand,  
That Caesar and my goodly father's world;

Write like Shakespeare

[\[reference\]](#)

↳ In reply to Thomas Paine



DeepDrumpf @DeepDrumpf · Mar 20

There will be no amnesty. It is going to pass because the people are  
going to be gone. I'm giving a mandate. #ComeyHearing  
@Thomas1774Paine

↳ 1

↪ 12

❤ 17

.. and Trump

[\[reference\]](#)



---

# Applications of RNNs

Technically, an RNN models sequences

Time series

Natural Language, Speech

We can even convert non-sequences to sequences, eg: feed an image as a sequence of pixels!

---



---

# Applications of RNNs

RNN Generated TED Talks

[YouTube Link](#)

RNN Generated Eminem rapper

[RNN Shady](#)

RNN Generated Music

[Music Link](#)

---



---

# Why RNNs?

Can model sequences having variable length

**Efficient:** Weights shared across time-steps

**They work!**

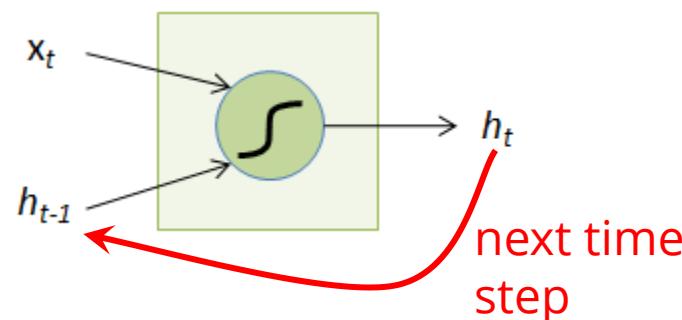
SOTA in several speech, NLP tasks



---

# The Recurrent Neuron

- $x_t$ : Input at time t
- $h_{t-1}$ : State at time t-1



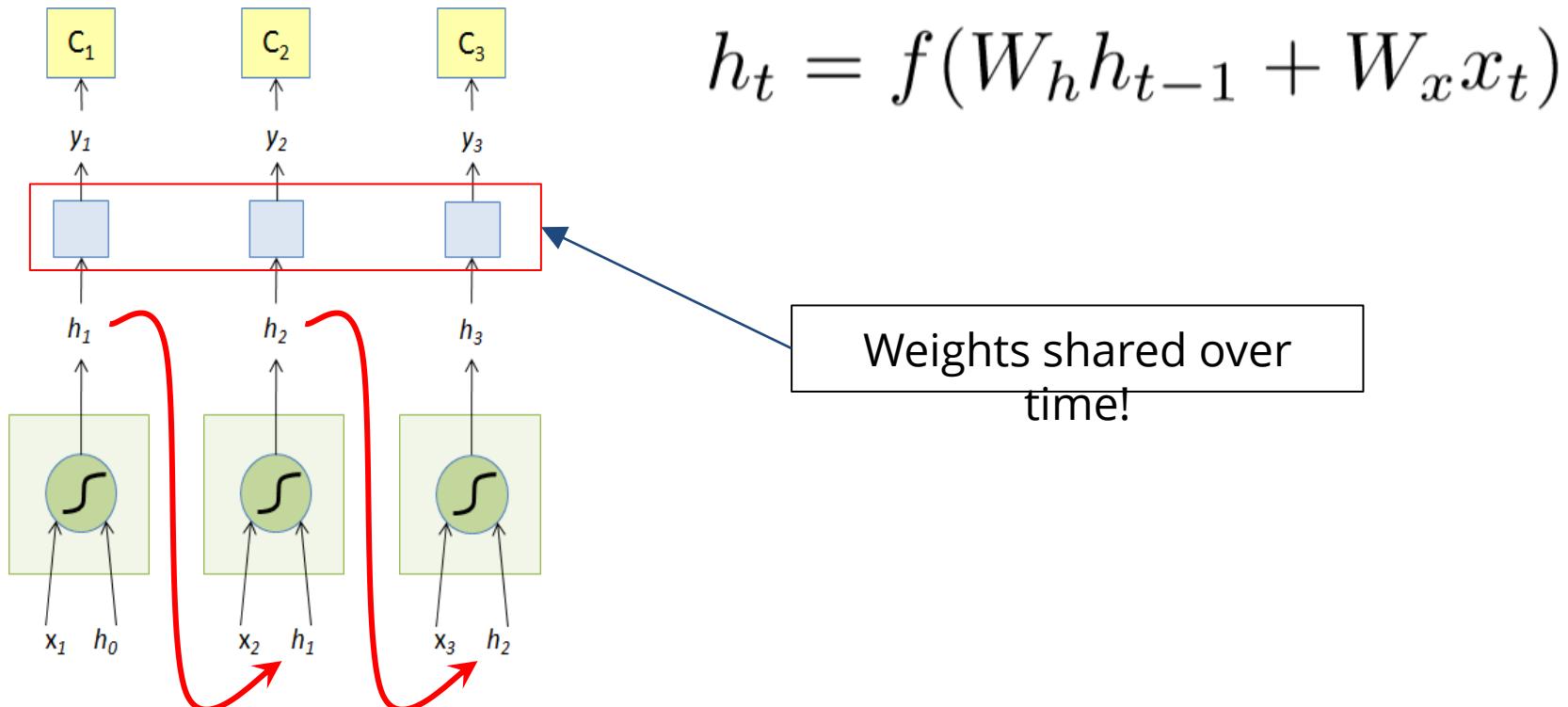
$$h_t = f(W_h h_{t-1} + W_x x_t)$$

Source: [Slides by Arun](#)

---



# Unfolding an RNN

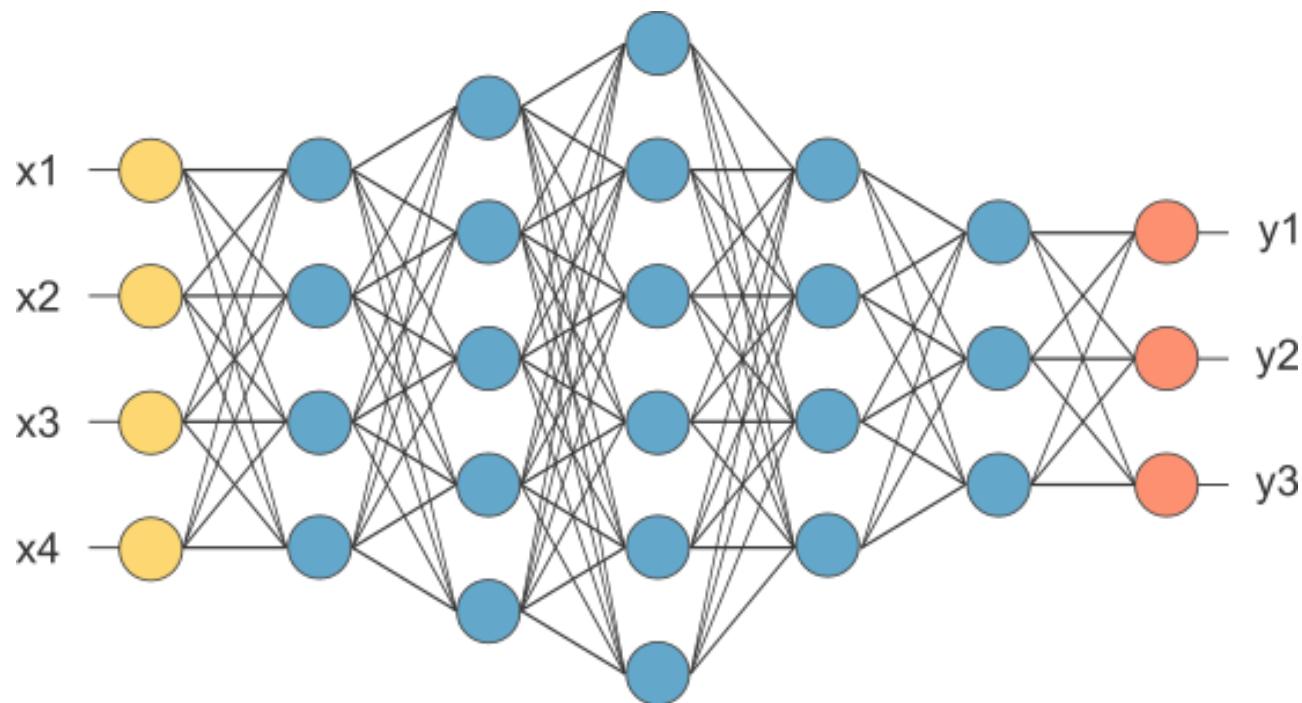


Source: [Slides by Arun](#)



---

# Making Feedforward Neural Networks Deep



Source: [http://www.opennn.net/images/deep\\_neural\\_network.png](http://www.opennn.net/images/deep_neural_network.png)

---

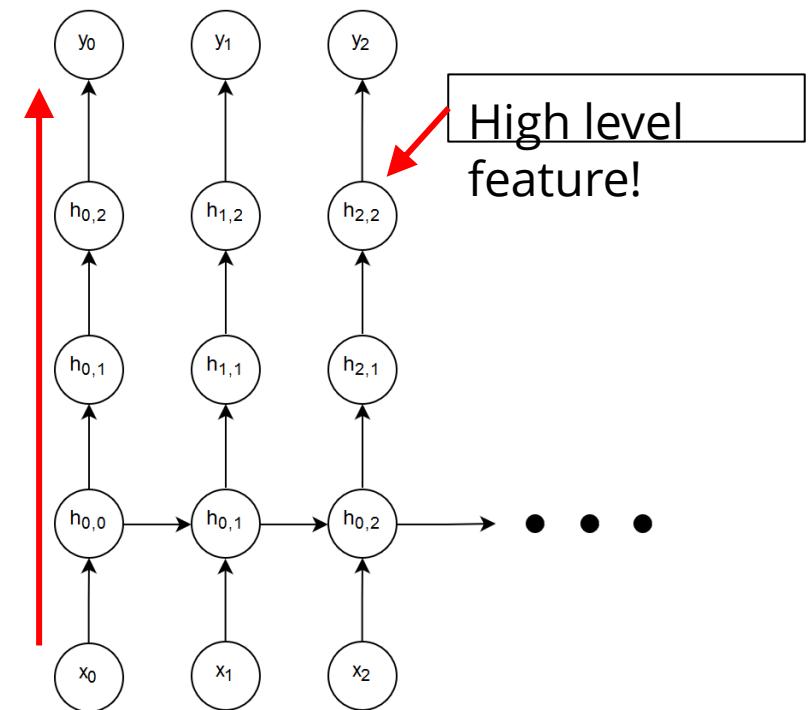


# Option 1: Feedforward Depth ( $d_f$ )

Notation:  $h_{0,1} \Rightarrow$  time step 0, neuron #1

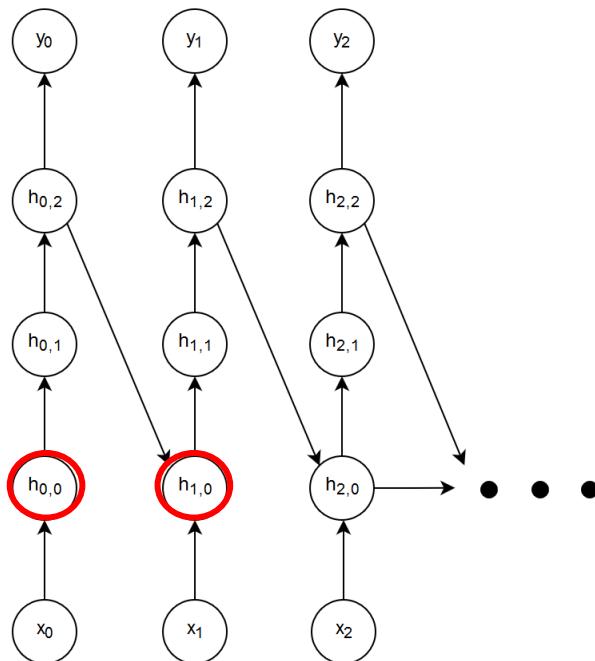
**Feedforward depth:** longest path  
between an input and output at the  
**same timestep**

Feedforward depth = 4





## Option 2: Recurrent Depth ( $d_r$ )

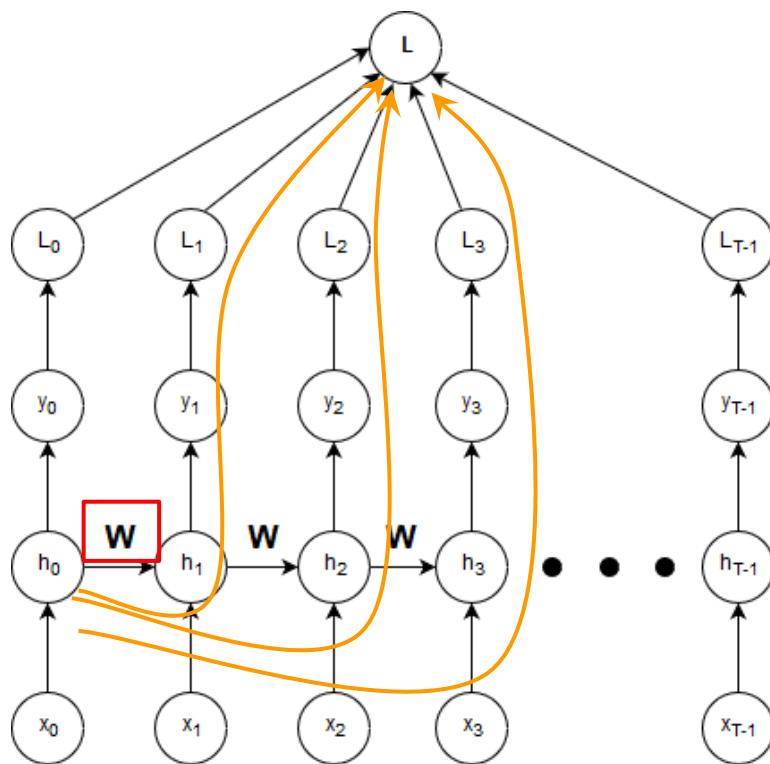


- Recurrent depth: Longest path between same hidden state in successive timesteps

Recurrent depth = 3



# Backpropagation Through Time (BPTT)



Objective is to update the weight matrix:

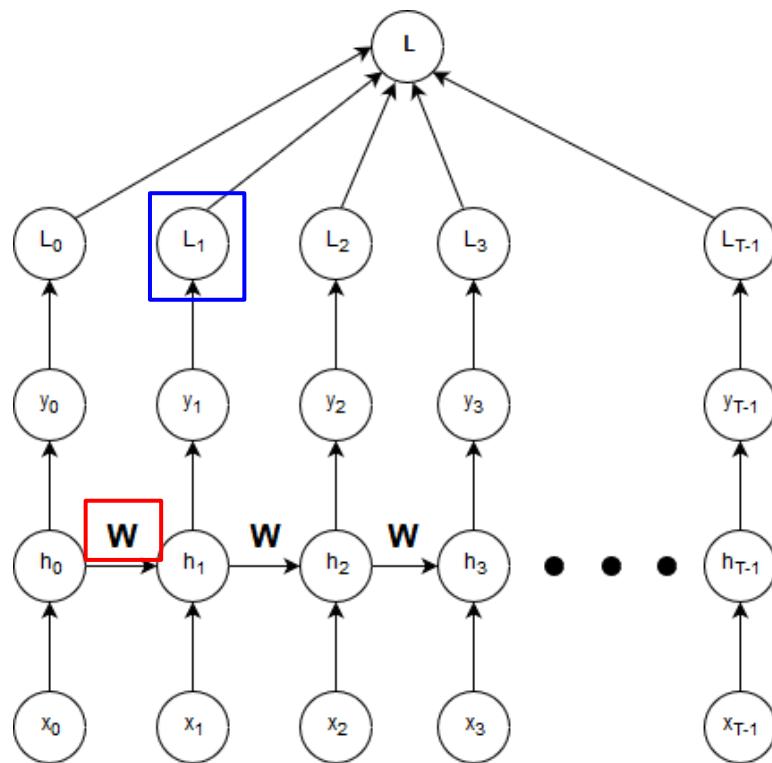
$$\mathbf{W} \rightarrow \mathbf{W} - \alpha \frac{\partial L}{\partial \mathbf{W}}$$

Issue:  $\mathbf{W}$  occurs each timestep  
**Every** path from  $\mathbf{W}$  to  $L$  is one dependency  
Find all paths from  $\mathbf{W}$  to  $L$ !

(note: dropping subscript  $h$  from  $\mathbf{W}_h$  for brevity)



# Systematically Finding All Paths

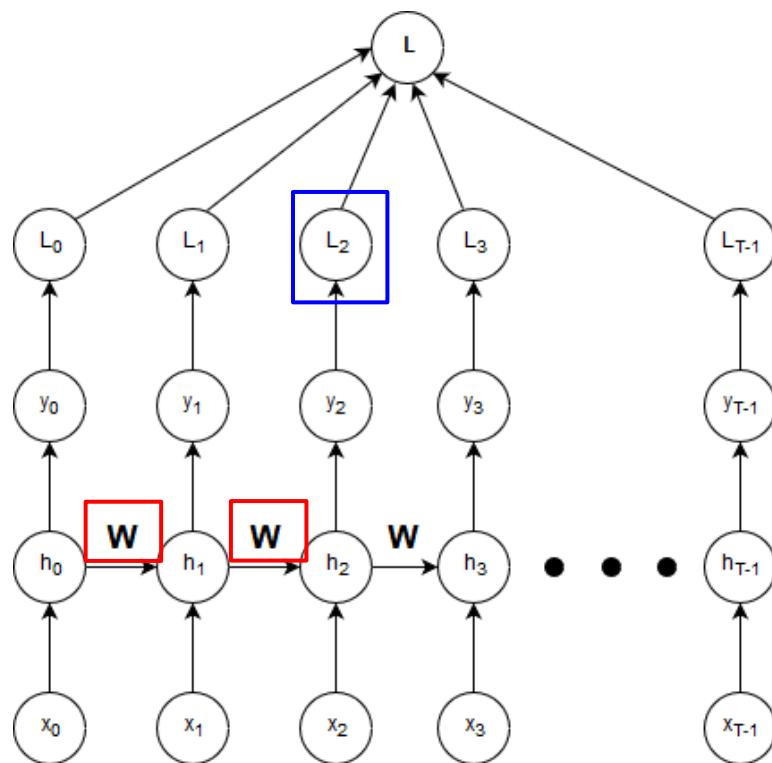


How many paths exist from  $W$  to  $L$  through  $L_1$ ?

Just 1. Originating at  $h_0$ .



# Systematically Finding All Paths

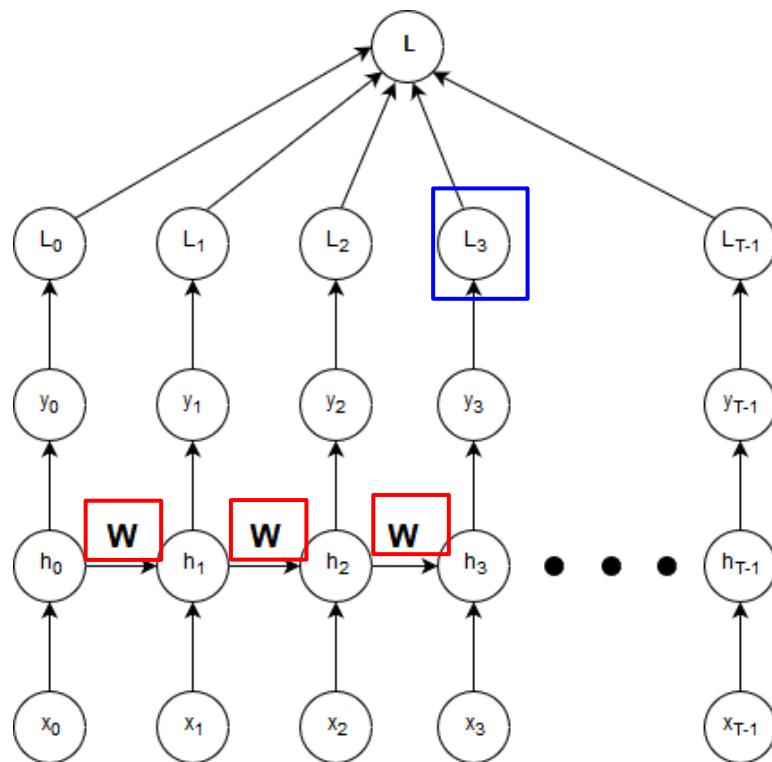


How many paths from  $W$  to  $L$  through  $L_2$ ?

2. Originating at  $h_0$  and  $h_1$ .



# Systematically Finding All Paths



And 3 in this case.

Origin of path = basis for  $\Sigma$

$$\frac{\partial L}{\partial \mathbf{W}}$$

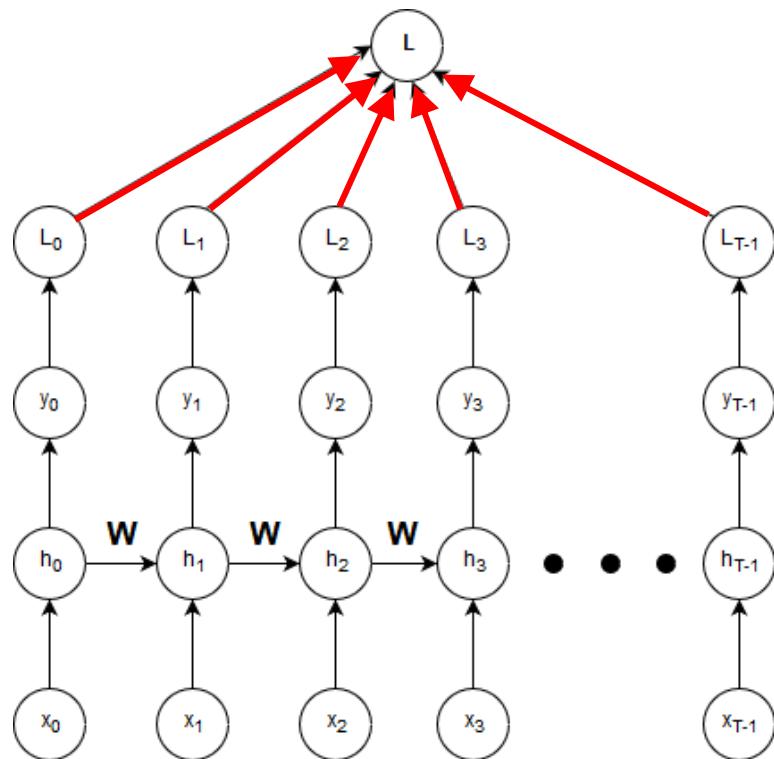
The gradient has two summations:  
1: Over  $L_j$   
2: Over  $h_k$

To skip proof, click [here](#).



---

# Backpropagation as two summations

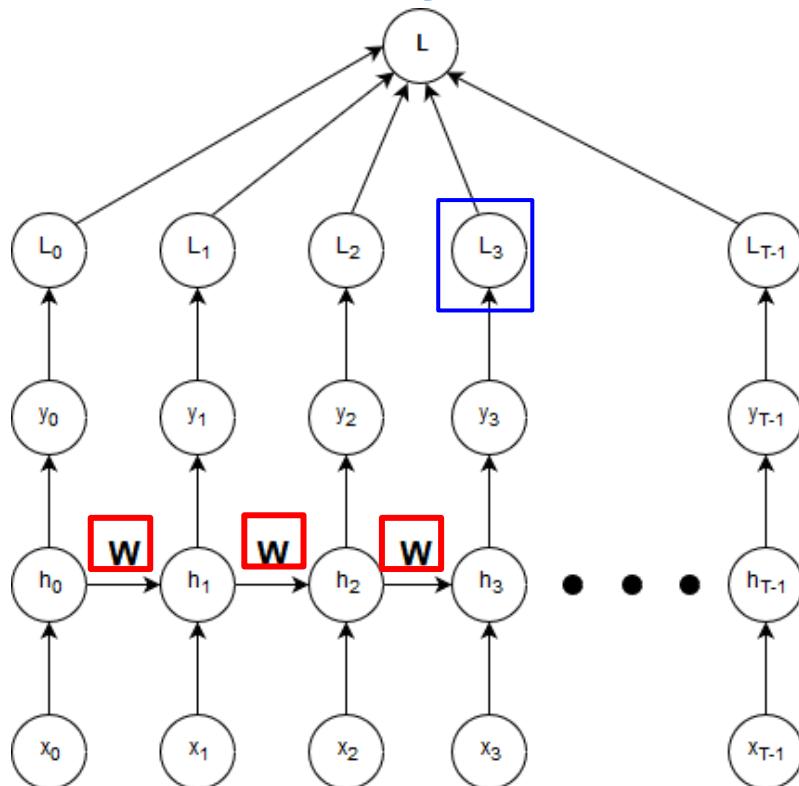


First summation over  $L$

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial \mathbf{W}}$$



# Backpropagation as two summations



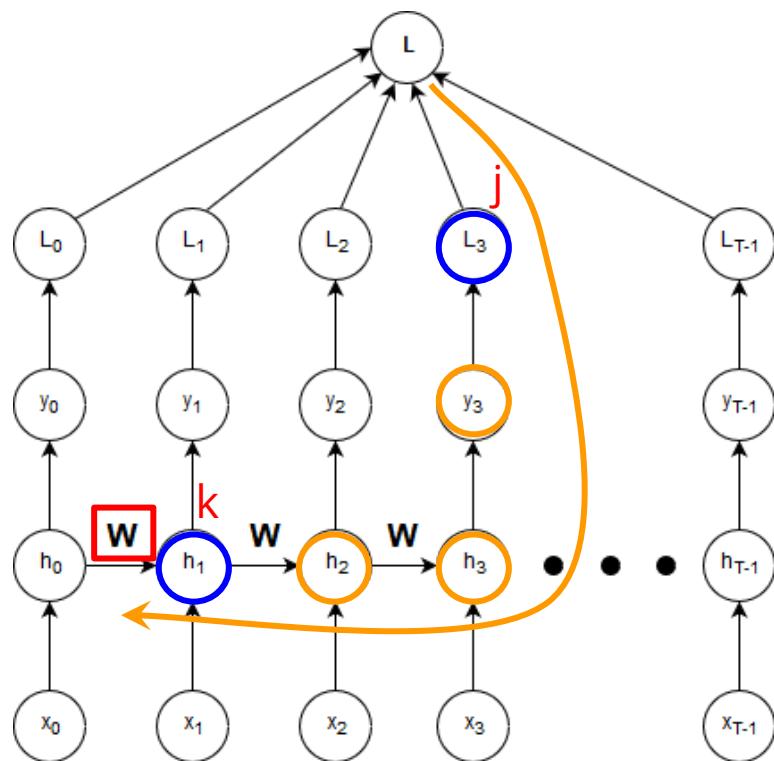
- Second summation over  $h$ :  
Each  $L_j$  depends on the weight matrices *before it*

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^j \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

$L_j$  depends on all  $h_k$  before it.



# Backpropagation as two summations

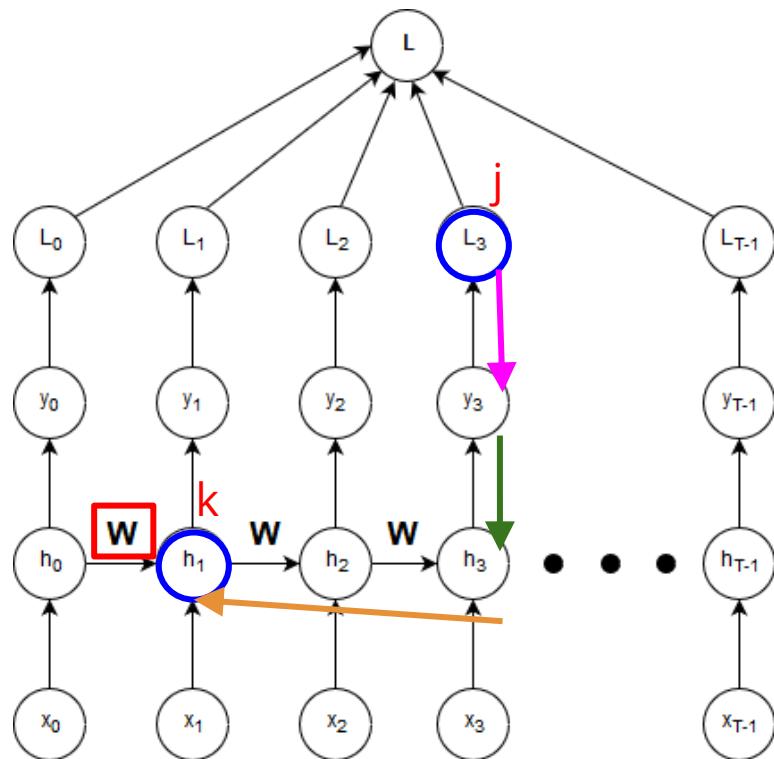


- $$\frac{\partial L_j}{\partial W} = \sum_{k=1}^j \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial W}$$
- No explicit of  $L_j$  on  $h_k$
  - Use chain rule to fill missing steps

$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial W}$$



# Backpropagation as two summations

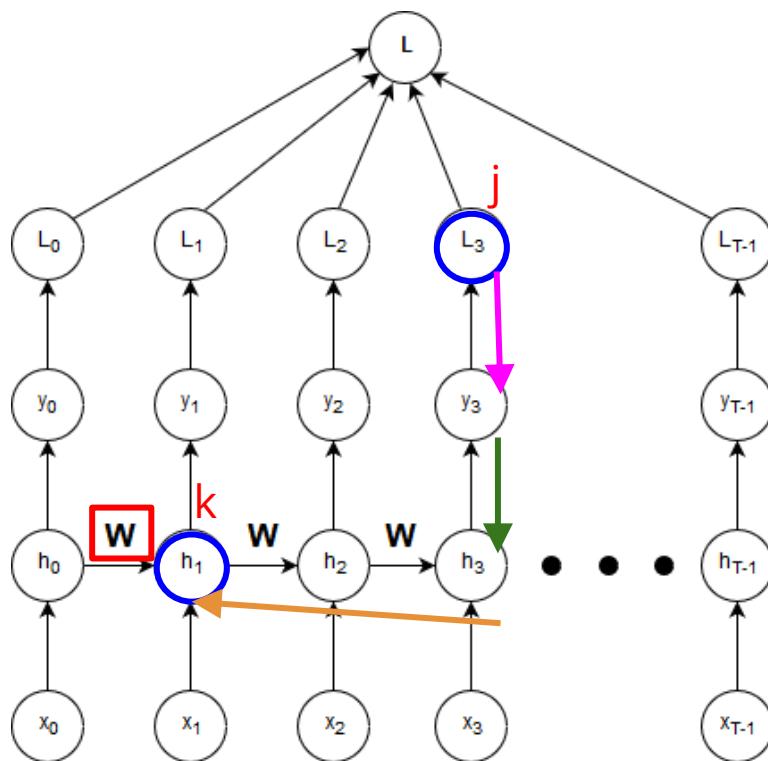


- $$\frac{\partial L_j}{\partial W} = \sum_{k=1}^j \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial W}$$
- No explicit of  $L_j$  on  $h_k$
  - Use chain rule to fill missing steps

$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial W}$$



# The Jacobian



$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^j \left[ \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \right] \frac{\partial h_k}{\partial \mathbf{W}}$$

Indirect dependency. One final use of the chain rule gives:

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^j \frac{\partial h_m}{\partial h_{m-1}}$$

“The Jacobian”



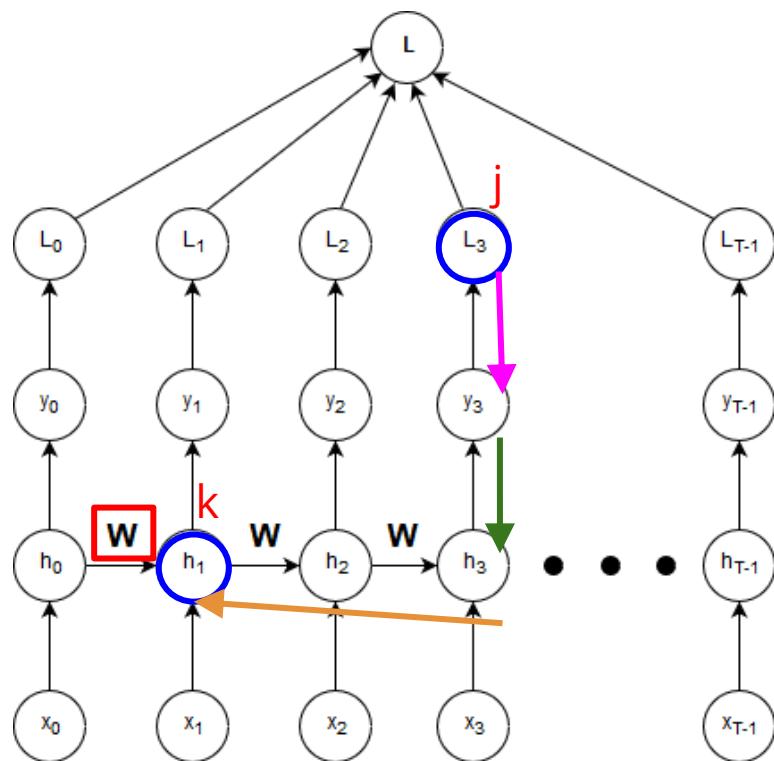
---

## The Final Backpropagation Equation

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{m=k+1}^j \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$



# Backpropagation as two summations



$$\frac{\partial L}{\partial \mathbf{W}_h} = \sum_{j=0}^{T-1} \sum_{k=1}^j \left[ \frac{\partial L_j}{\partial y_j} \right] \left[ \frac{\partial y_j}{\partial h_j} \right] \left( \prod_{m=k+1}^j \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}_h}$$

- Often, to reduce memory requirement, we truncate the network
- Inner summation runs from  $j-p$  to  $j$  for some  $p \Rightarrow$  truncated BPTT



---

## Expanding the Jacobian

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{m=k+1}^j \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \text{diag}(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m))$$



# The Issue with the Jacobian

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^j \mathbf{W}_h^T \text{diag}(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m))$$

Weight Matrix

Derivative of activation function

Repeated matrix multiplications leads to **vanishing and exploding gradients**.  
How? Let's take a slight detour.



---

# Eigenvalues and Stability

Consider identity activation function

If Recurrent Matrix  $\mathbf{W}_h$  is a diagonalizable:

$$\mathbf{W}_h = Q^{-1} * \Lambda * Q$$

Computing powers of  $\mathbf{W}_h$  is simple:

$$\mathbf{W}_h^n = Q^{-1} * \Lambda^n * Q$$

Q matrix composed of eigenvectors of  $\mathbf{W}_h$

$\Lambda$  is a diagonal matrix with eigenvalues placed on the diagonals

Bengio et al, "On the difficulty of training recurrent neural networks." (2012)

---



---

## Eigenvalues and stability

$$\Lambda = \begin{bmatrix} -0.6180 & 0 \\ 0 & 1.6180 \end{bmatrix}$$



Vanishing  
gradients

$$\Lambda^{10} = \begin{bmatrix} 0.0081 & 0 \\ 0 & 122.9919 \end{bmatrix}$$



Exploding  
gradients

$$W_h^n = Q^{-1} * \Lambda^n * Q$$



---

All Eigenvalues  $< 1$   
Eigenvalues  $> 1$

Blog on “Explaining and illustrating orthogonal initialization for recurrent  
neural network”

---



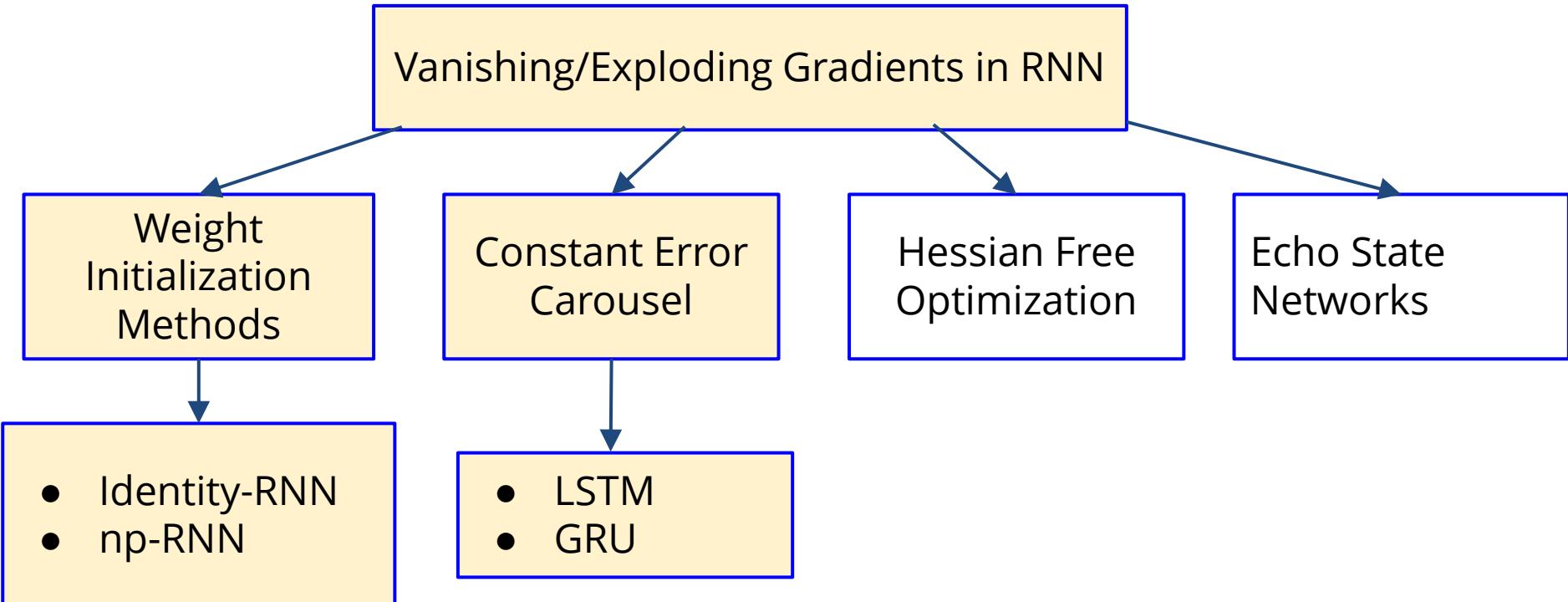
---

## 2. Learning Long Term Dependencies



---

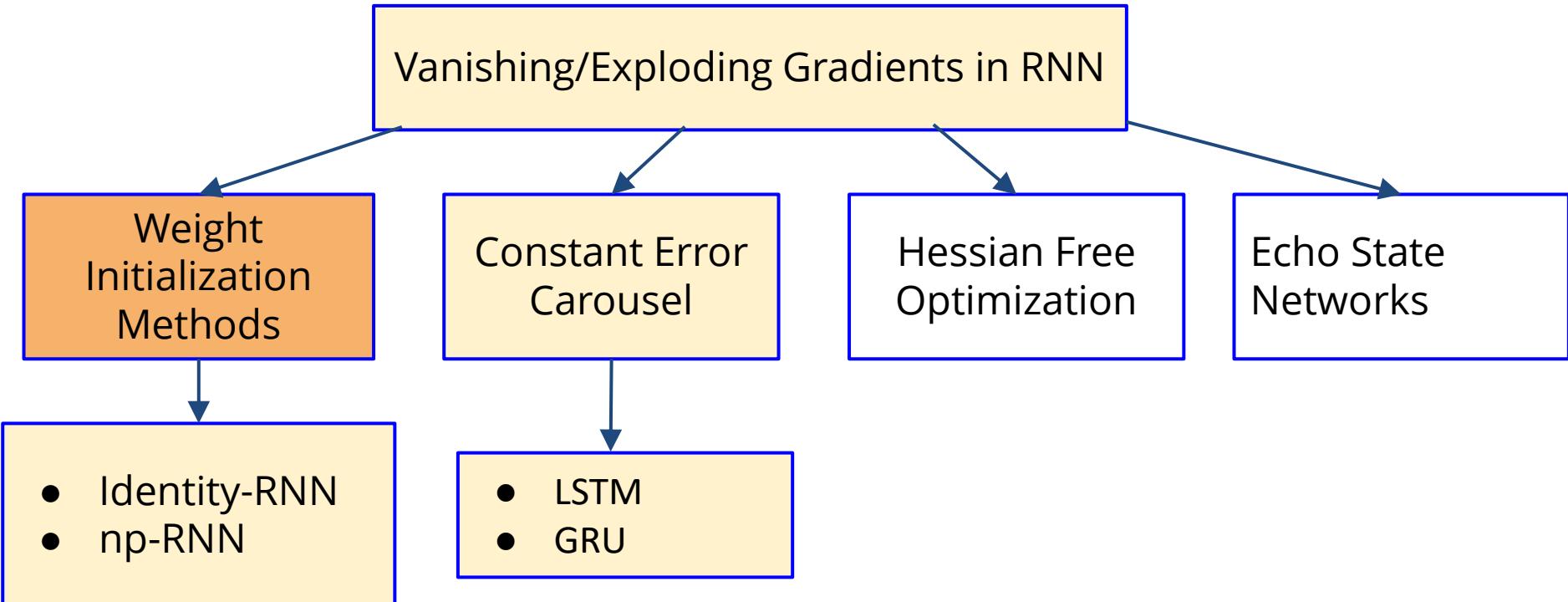
# Outline





---

# Outline





---

## Weight Initialization Methods

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^j \mathbf{W}_h^T \text{diag}(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m))$$

Activation function : ReLU

$$\frac{\partial h_j}{\partial h_k} = (\mathbf{W}_h^T)^n = Q^{-1} * \Lambda^n * Q$$

Bengio et al., "On the difficulty of training recurrent neural networks." (2012)

---



---

# Weight Initialization Methods

Random  $W_h$  initialization of RNN has no constraint on eigenvalues

⇒ vanishing or exploding gradients in the initial epoch



---

# Weight Initialization Methods

Careful initialization of  $W_h$  with suitable eigenvalues

- ⇒ allows the RNN to learn in the initial epochs
  - ⇒ hence can generalize well for further iterations
-



---

## Weight Initialization Trick #1: IRNN

- $W_h$  initialized to Identity
- Activation function: ReLU

Geoffrey et al, "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units"

---



---

## Weight Initialization Trick #2: np-RNN

- $W_h$  positive definite (+ve real eigenvalues)
- At least one eigenvalue is 1, others all less than equal to one
- Activation function: ReLU

Geoffrey et al, "Improving Performance of Recurrent Neural Network with ReLU nonlinearity""

---



# np-RNN vs IRNN

Sequence Classification Task

RNN Type	Accuracy Test	Parameter Complexity Compared to RNN	Sensitivity to parameters
IRNN	67 %	x1	high
np-RNN	75.2 %	x1	low
LSTM	78.5 %	x4	low

Geoffrey et al, "Improving Performance of Recurrent Neural Network with ReLU nonlinearity""



---

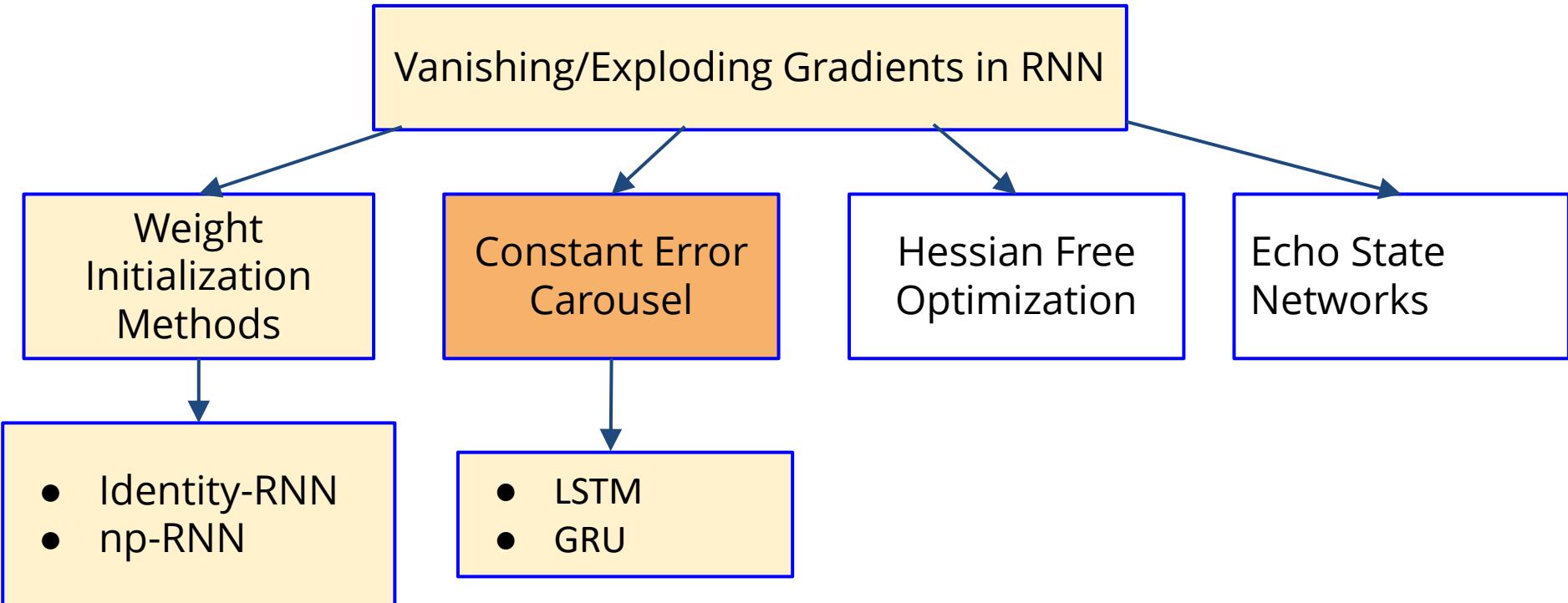
# Summary

np-RNNs work as well as LSTMs utilizing 4 times less parameters than a LSTM



---

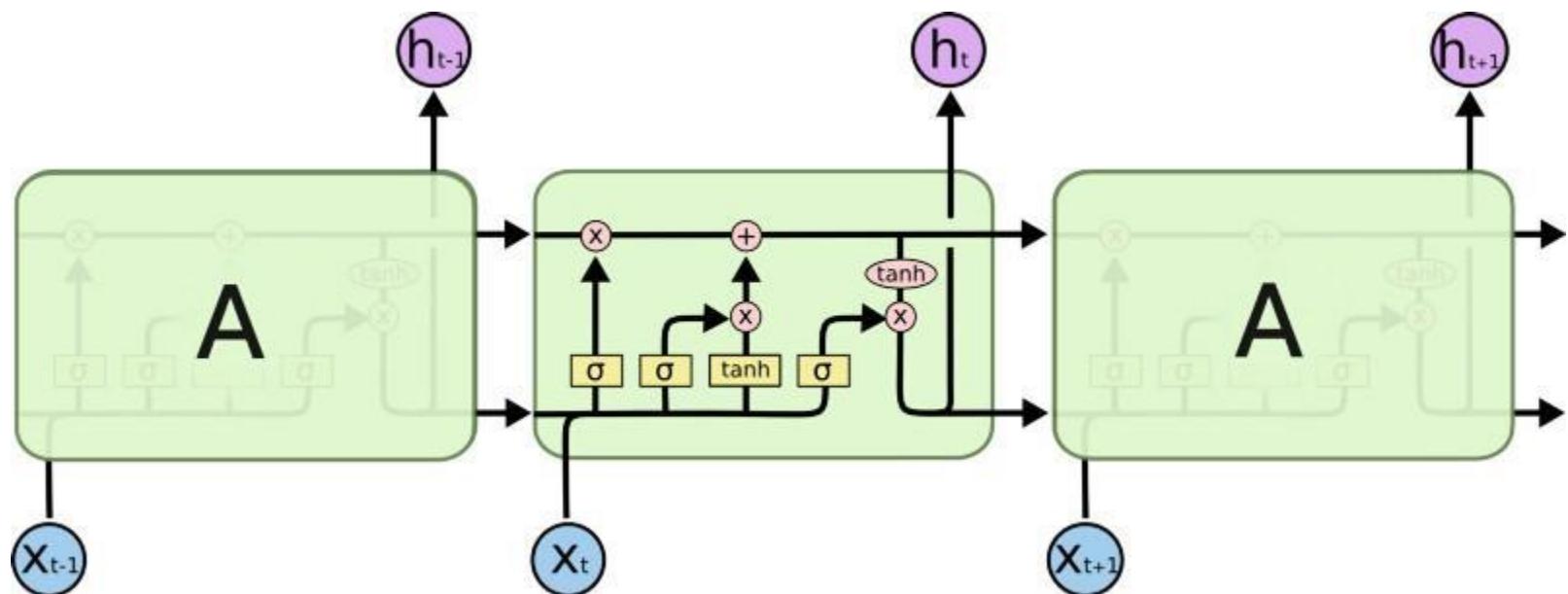
# Outline





---

# The LSTM Network

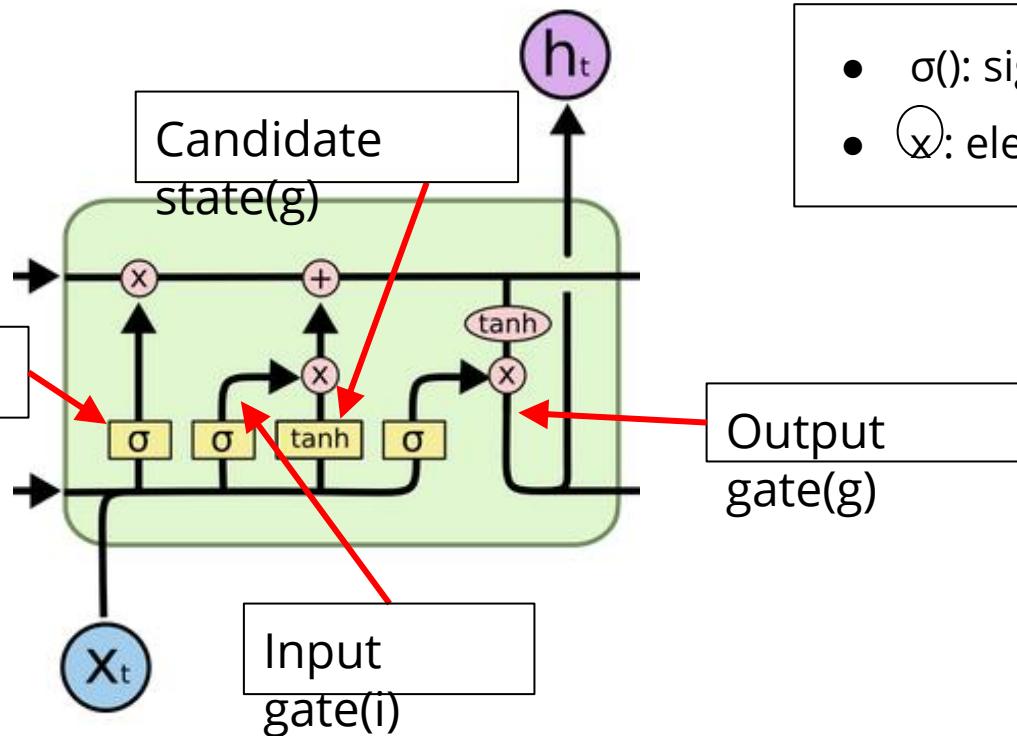


Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

---

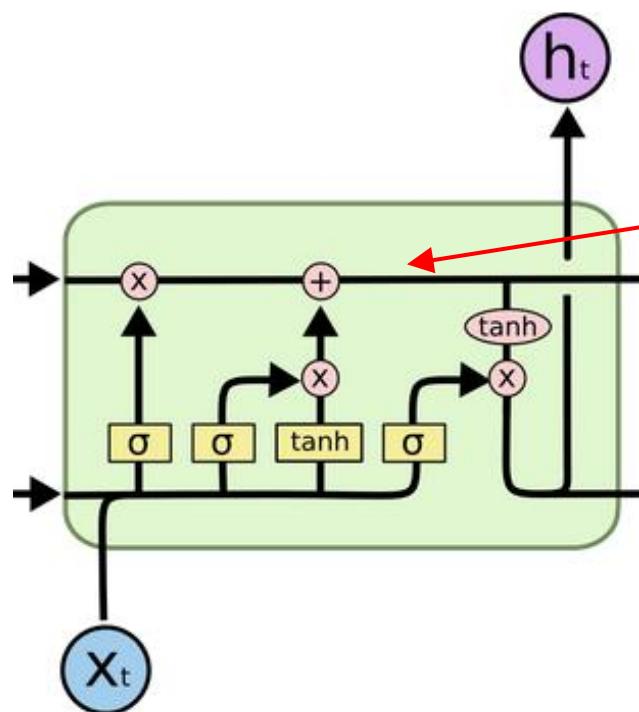


# The LSTM Cell





# The LSTM Cell



$$\begin{pmatrix} \tilde{f}_t \\ \tilde{i}_t \\ \tilde{o}_t \\ \tilde{g}_t \end{pmatrix} = \mathbf{W}_h h_{t-1} + \mathbf{W}_x x_t + \mathbf{b}$$
$$c_t = \sigma(\tilde{f}_t) \odot c_{t-1} + \sigma(\tilde{i}_t) \odot \tanh(\tilde{g}_t)$$
$$h_t = \sigma(o_t) \odot \tanh(c_t),$$

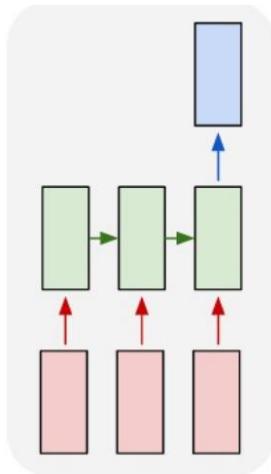
Forget old state

Remember new state



# Long Term Dependencies with LSTM

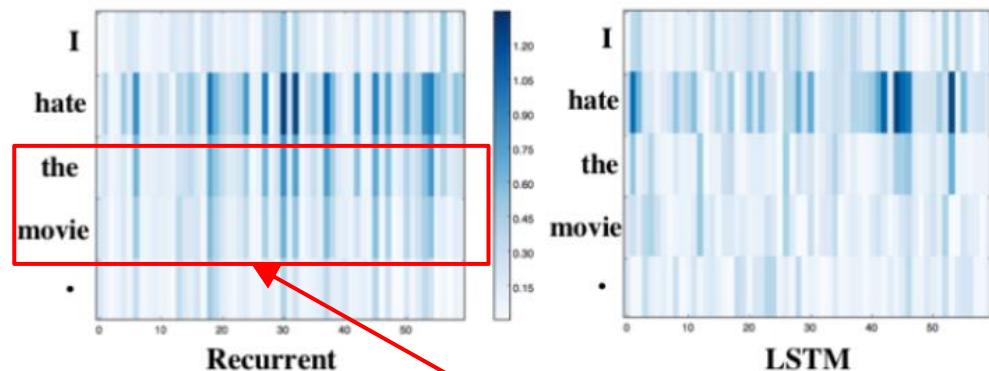
## Sentiment Analysis



Many-one

"Jiwei LI et al, "Visualizing and Understanding Neural ures long term Models in NLP"

## Saliency Heatmap

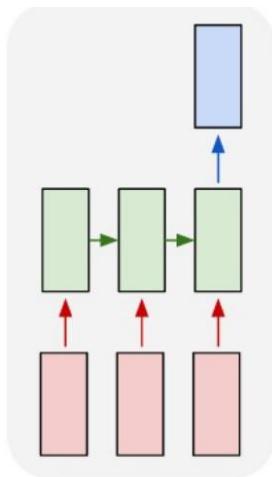


Recent words more  
salient



# Long Term Dependencies with LSTM

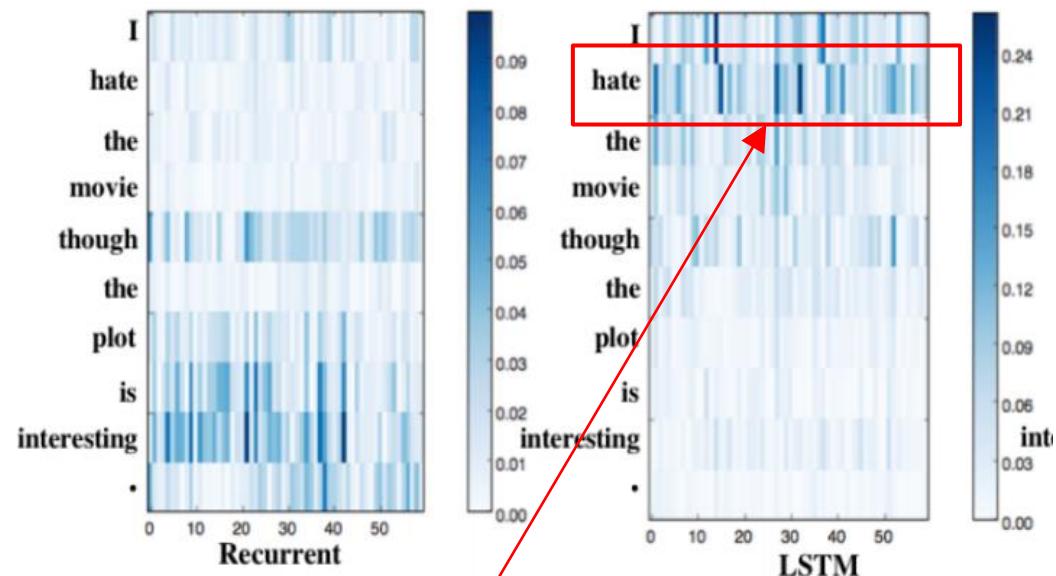
## Sentiment Analysis



Many-one

"Jiwei LI et al, "Visualizing and Understanding Neural ures long term Models in NLP"

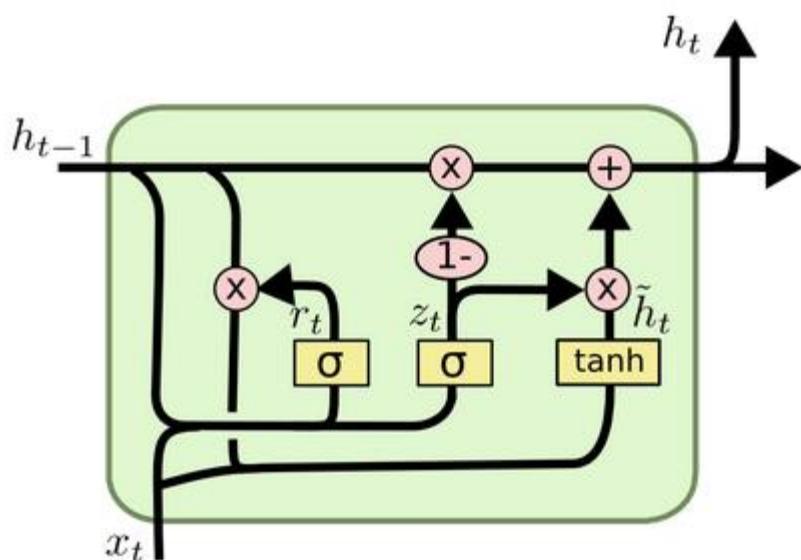
## Saliency Heatmap



uses long term  
dependencies



# Gated Recurrent Unit



- Replace forget (f) and input (i) gates with an update gate (z)
- Introduce a reset gate (r ) that modifies  $h_{t-1}$
- Eliminate internal memory  $c_t$

Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>



---

## Comparing GRU and LSTM

Both **GRU** and **LSTM** better than **RNN** with **tanh** on music and speech modeling

GRU performs comparably to LSTM

No clear consensus between GRU and LSTM

Source: Empirical evaluation of GRUs on sequence modeling, 2014

---



---

### 3. Regularization in RNNs



---

# Outline

Batch Normalization

Dropout

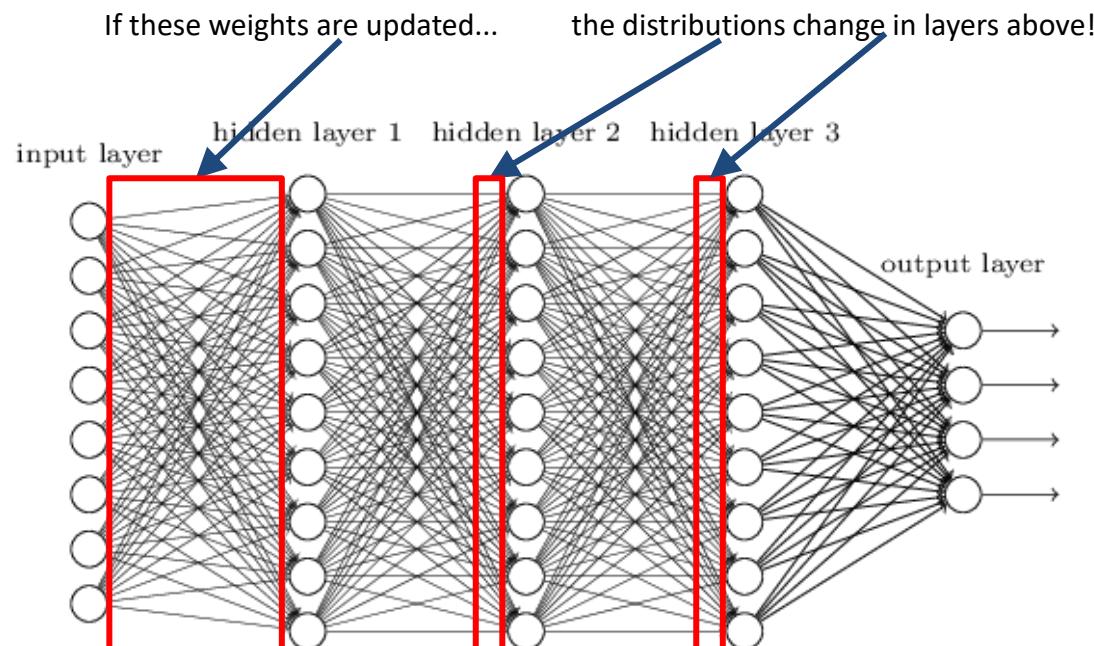


---

# Recurrent Batch Normalization



# Internal Covariate Shift



The model needs to learn parameters while **adapting to the changing input distribution**  
⇒ slower model convergence!

Source: <https://i.stack.imgur.com/1bCQI.png>

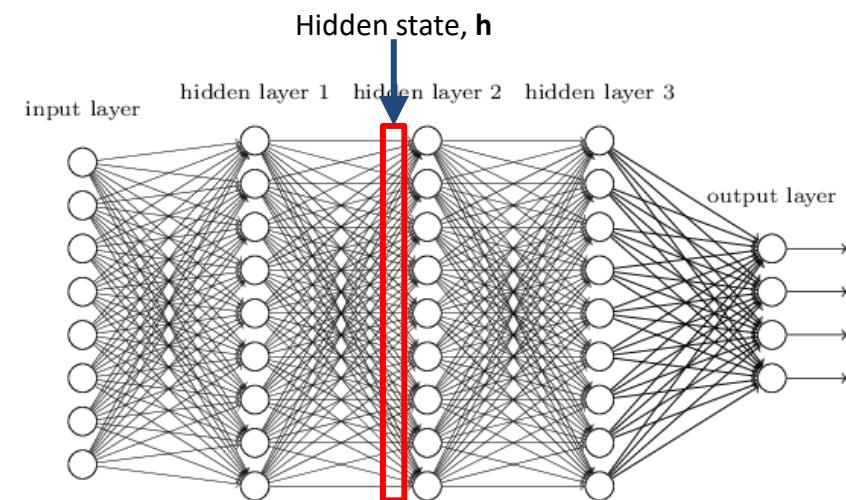


# Solution: Batch Normalization

Batch Normalization Equation:

$$BN(\mathbf{h}; \gamma, \beta) = \beta + \gamma \odot \frac{\mathbf{h} - \hat{\mathbb{E}}[\mathbf{h}]}{\sqrt{\text{Var}[\mathbf{h}] + \epsilon}}$$

Bias, Std Dev: To be learned



Cooijmans, Tim, et al. "Recurrent batch normalization."(2016).



---

## Extension of BN to RNNs: Trivial?

RNNs **deepest along temporal dimension**

Must be careful: repeated scaling could cause **exploding gradients**



---

# The method that's effective

$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}$$

$$\begin{aligned} \mathbf{c}_t &= \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t) \\ \mathbf{h}_t &= \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\mathbf{c}_t), \end{aligned}$$

**Original LSTM Equations**

$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \text{BN}(\mathbf{W}_h \mathbf{h}_{t-1}; \gamma_h, \beta_h) + \text{BN}(\mathbf{W}_x \mathbf{x}_t; \gamma_x, \beta_x) + \mathbf{b}$$

$$\begin{aligned} \mathbf{c}_t &= \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t) \\ \mathbf{h}_t &= \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\text{BN}(\mathbf{c}_t; \gamma_c, \beta_c)) \end{aligned}$$

**Batch Normalized LSTM**

Cooijmans, Tim, et al. "Recurrent batch normalization."(2016).

---



# Observations

- $x, h_{t-1}$  normalized **separately**

- $c_t$  **not normalized**

(doing so may disrupt  
gradient flow) **How?**

- New state ( $h_t$ ) normalized

$$\begin{pmatrix} \tilde{f}_t \\ \tilde{i}_t \\ \tilde{o}_t \\ \tilde{g}_t \end{pmatrix} = \boxed{\text{BN}(\mathbf{W}_h h_{t-1}; \gamma_h, \beta_h)} + \boxed{\text{BN}(\mathbf{W}_x x_t; \gamma_x, \beta_x)} + \mathbf{b}$$
$$c_t = \sigma(\tilde{f}_t) \odot c_{t-1} + \sigma(\tilde{i}_t) \odot \tanh(\tilde{g}_t)$$
$$h_t = \sigma(\tilde{o}_t) \odot \tanh(\text{BN}(c_t; \gamma_c, \beta_c))$$

Cooijmans, Tim, et al. "Recurrent batch normalization."(2016).



---

## Additional Guidelines

$$\text{BN}(\mathbf{W}_h \mathbf{h}_{t-1}; \gamma_h, \beta_h)$$

- Initialize  $\beta$  to 0,  $\gamma$  to a small value such as  $\sim 0.1$ . Else vanishing gradients (think of the tanh plot!)

Learn statistics for each time step **independently** till some time step **T**.

Beyond **T**, use statistics for **T**

Cooijmans, Tim, et al. "Recurrent batch normalization."(2016).

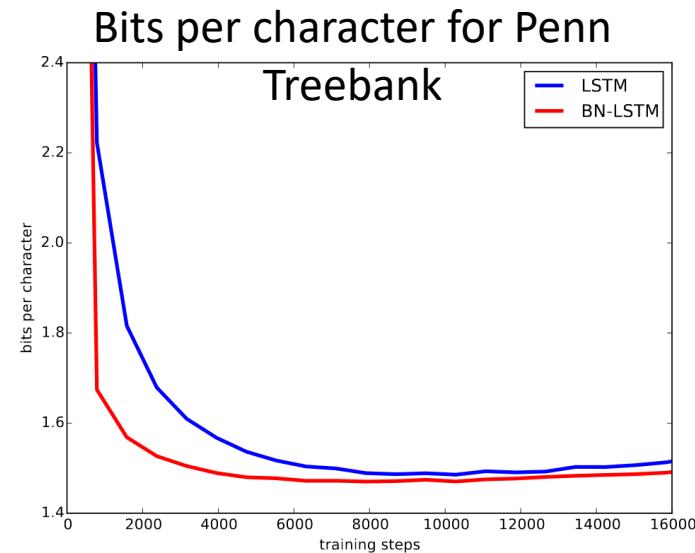
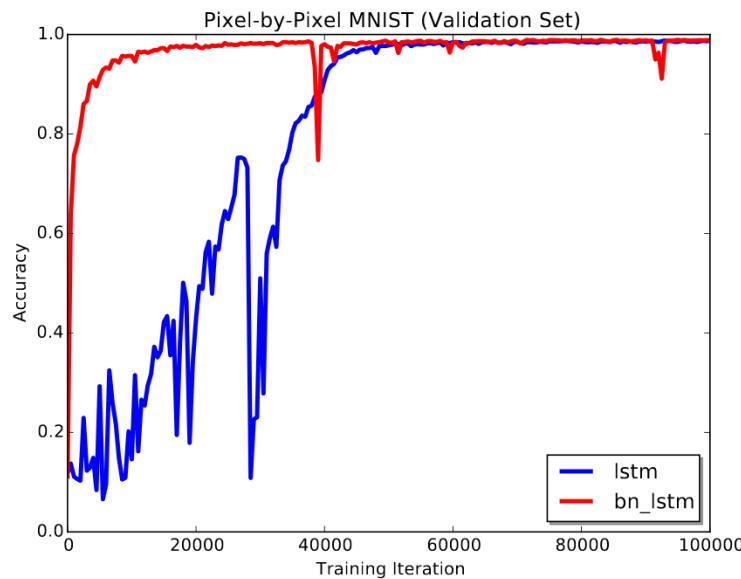
---



# Results

A: **Faster convergence** due to Batch Norm

B: Performance **as good** as (if not better than) unnormalized LSTM



Cooijmans, Tim, et al. "Recurrent batch normalization."(2016).



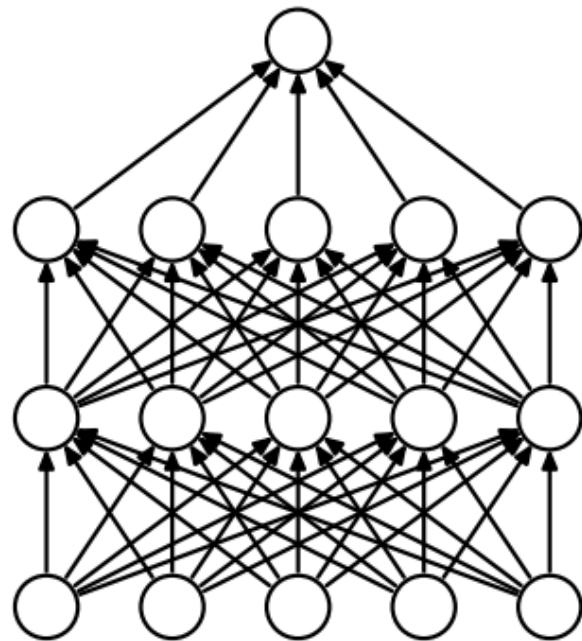
---

# Dropout In RNN

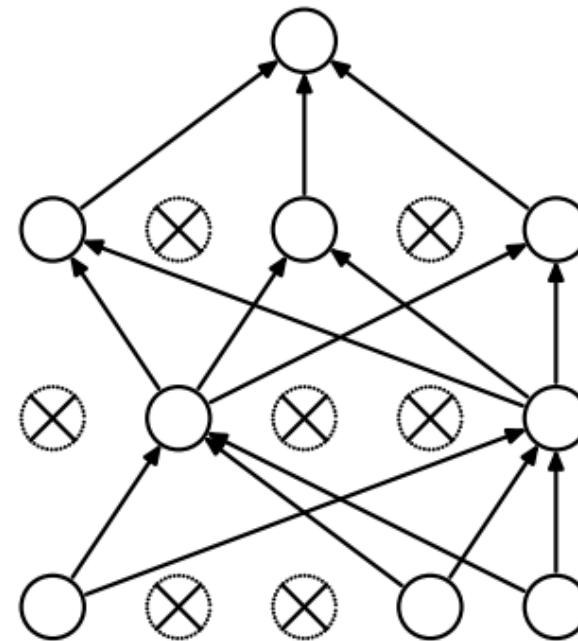


---

## Recap: Dropout In Neural Networks



(a) Standard Neural Net



(b) After applying dropout.

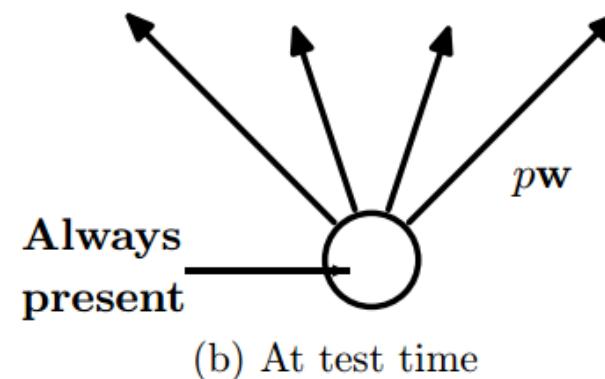
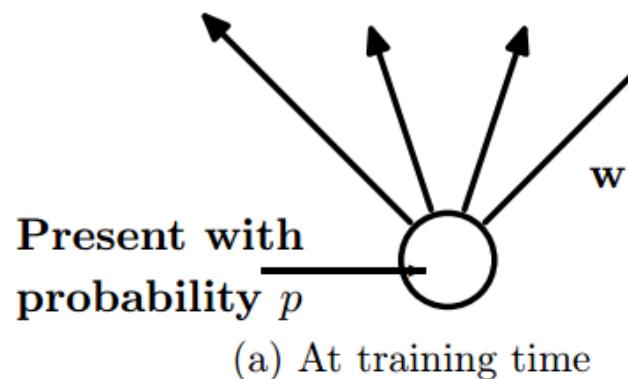
Srivastava et al. 2014. "Dropout: a simple way to prevent neural networks from overfitting"

---



---

## Recap: Dropout In Neural Networks



Srivastava et al. 2014. "Dropout: a simple way to prevent neural networks from overfitting"

---



---

# Dropout

To prevent over confident models

High Level Intuition: Ensemble of thinned networks sampled through dropout

Interested in a theoretical proof ?

A Probabilistic Theory of Deep Learning, [Ankit B. Patel](#), Tan Nguyen, [Richard G. Baraniuk](#)

[Skip Proof Slides](#)

---

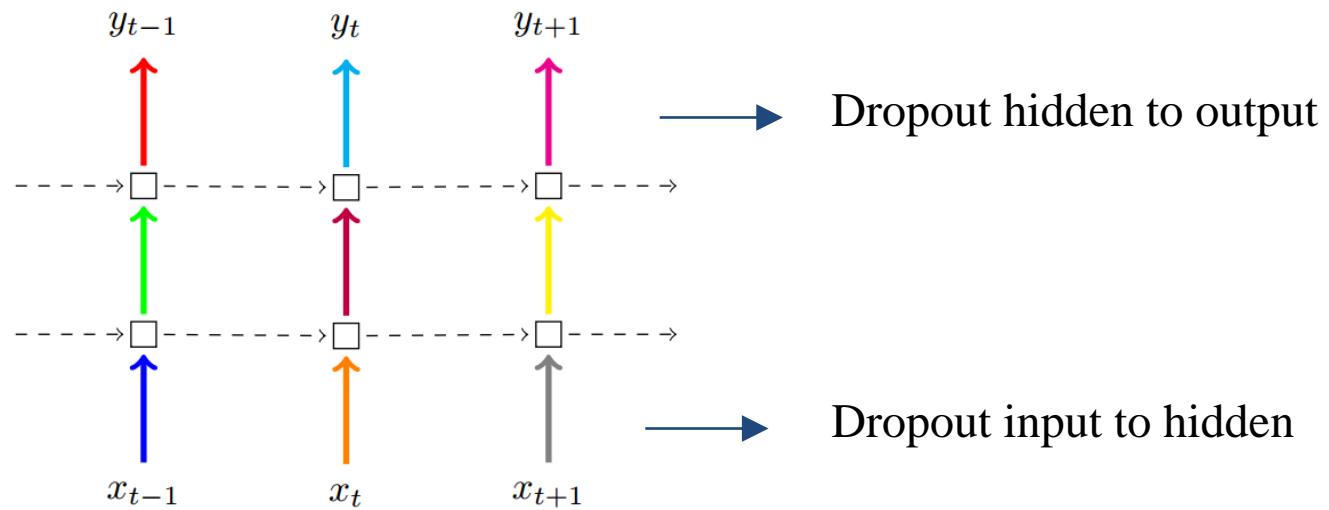


# RNN Feedforward Dropout

Beneficial to use it once in correct spot rather than put it everywhere

Each color represents a different mask

Per-step mask sampling

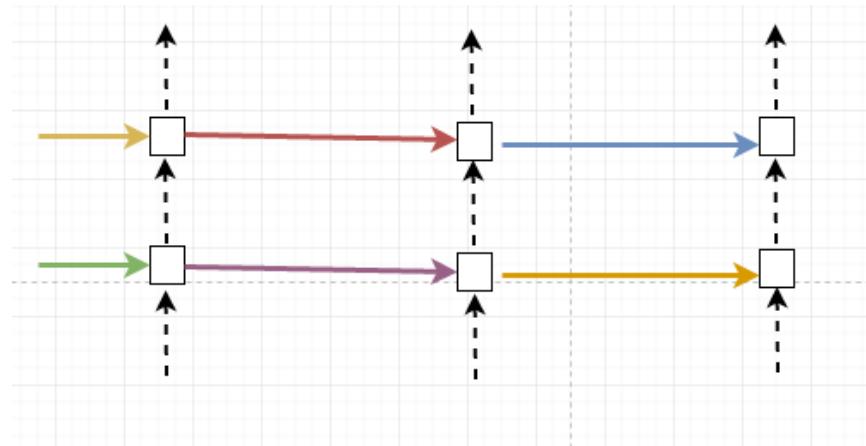


Zaremba et al. 2014. "Recurrent neural network regularization"



---

# RNN Recurrent Dropout

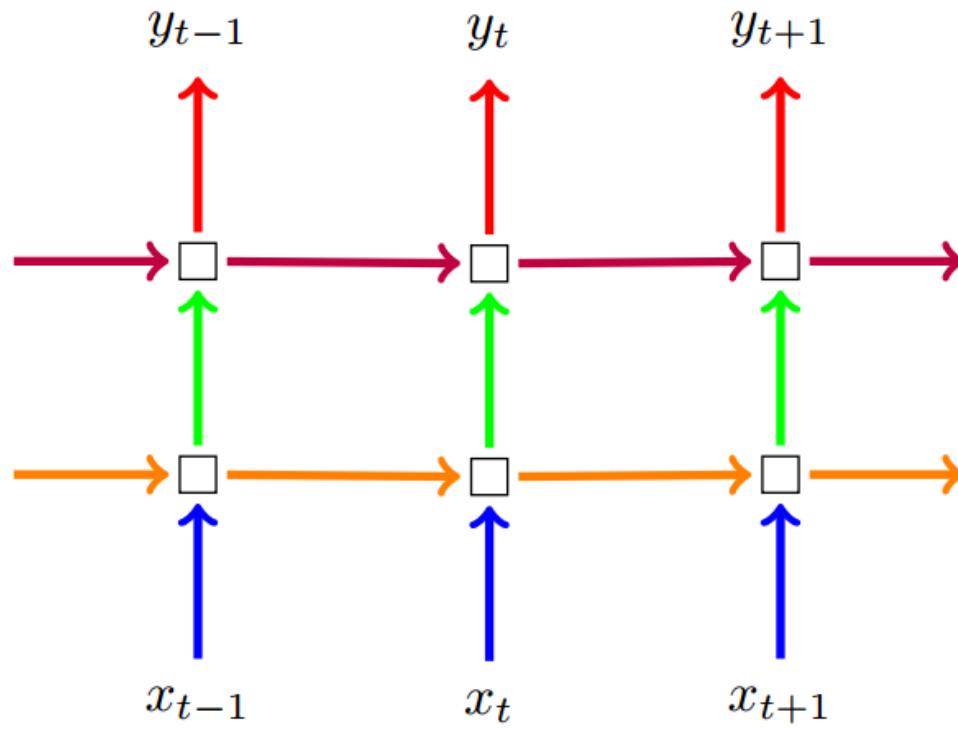


MEMORY LOSS !  
Only tends to retain short term dependencies

---



# RNN Recurrent+Feedforward Dropout



Per-sequence mask sampling

Drop the time dependency of  
an entire feature

Gal 2015. "A theoretically grounded application of dropout in recurrent neural networks"



---

# Dropout in LSTMs

Dropout on cell state ( $c_t$ )

Inefficient

Dropout on cell state update  
( $\tanh(g)_t$ ) or ( $h_{t-1}$ )

Optimal

$$\begin{pmatrix} \tilde{f}_t \\ \tilde{i}_t \\ \tilde{o}_t \\ \tilde{g}_t \end{pmatrix} = \mathbf{W}_h h_{t-1} + \mathbf{W}_x x_t + \mathbf{b}$$
$$c_t = \sigma(\tilde{f}_t) \odot c_{t-1} + \sigma(\tilde{i}_t) \odot \tanh(\tilde{g}_t)$$
$$h_t = \sigma(\tilde{o}_t) \odot \tanh(c_t),$$

[Skip to Visualization](#)

Barth (2016) : "Semenuita et al. 2016. "Recurrent dropout without memory loss"

---



---

## Some Results: Language Modelling Task

Model	Perplexity Scores
Original	125.2
Forward Dropout + Drop ( $\tanh(g_t)$ )	87 (-37)
Forward Dropout + Drop ( $h_{t-1}$ )	88.4 (-36)
Forward Dropout	89.5 (-35)
Forward Dropout + Drop ( $c_t$ )	99.9 (-25)

Lower perplexity score is better !

Barth (2016) : "Semeniuta et al. 2016. "Recurrent dropout without memory loss"

---



---

## Section 4: Visualizing and Understanding Recurrent Networks



---

# Visualization outline

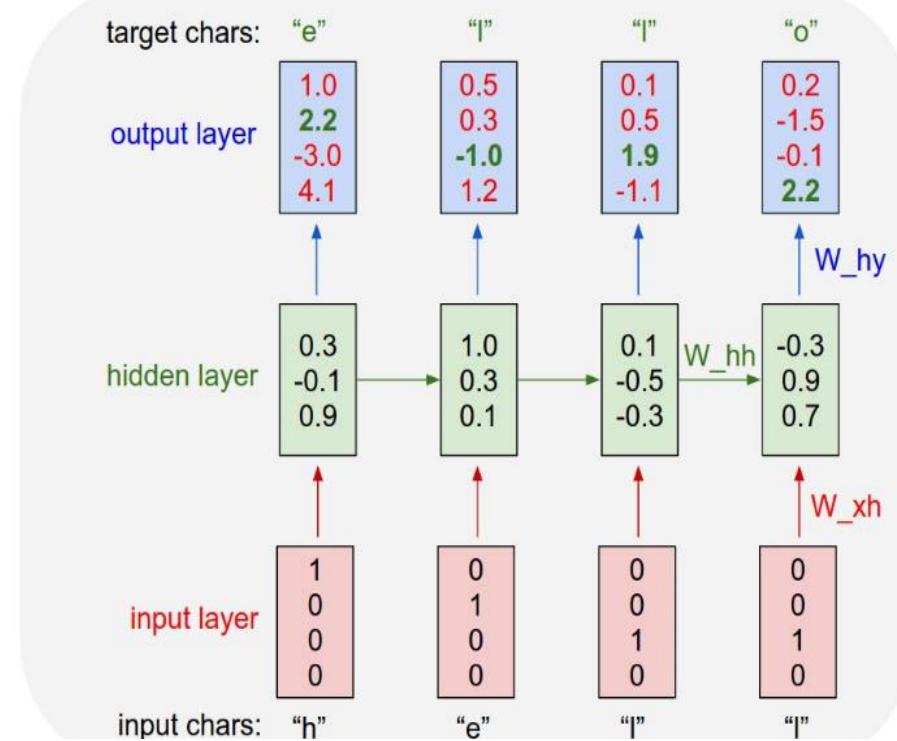
Observe evolution of features during training  
Visualize output predictions  
Visualize neuron activations

} Character Level  
Language Modelling  
task



# Character Level Language Modelling

**Task:** Predicting the next character given the current character



Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”



## Generated Text:

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servitious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (P?)<http://www.humah.yahoo.com/guardian.cfm/7754800786d1/551963589.htm> Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]

Andrej Karpathy, Blog on "Unreasonable Effectiveness of Recurrent Neural Networks"

- Remembers to close a bracket
- Capitalize nouns
- 404 Page Not Found! :P The LSTM



## 100 th iteration

tyntd-iafhatawiaoahrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

↓ train more

## 300 th iteration

"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwyl fil on aseterlome  
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

↓ train more

## 700 th iteration

Aftair fall unsuch that the hall for Prince Velzonski's that me of  
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort  
how, and Gogition is so overelical and ofter.

## 2000 th iteration

↓ train more

"Why do what that day," replied Natasha, and wishing to himself the fact the  
princess, Princess Mary was easier, fed in had oftened him.

Pierre akinng his soul came to the backs and drove un his father-in-law women.

Andrej Karpathy, Blog on "Unreasonable Effectiveness of Recurrent Neural Networks"



# Visualizing Predictions and Neuron “firings”



Likely prediction  
 Not a likely prediction

Excited neuron in  
url  
 Not excited neuron  
outside url

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”



---

# Features RNN Captures in Common Language ?

"Why do what that day," replied Natasha, and wishing to himself the fact the princess, Princess Mary was easier, fed in had oftened him.  
Pierre aking his soul came to the packs and drove up his father-in-law women.





---

## Cell Sensitive to Position in Line

- Can be interpreted as tracking the line length

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

Andrej Karpathy, Blog on "Unreasonable Effectiveness of Recurrent Neural Networks"

---



---

## Cell That Turns On Inside Quotes

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Andrej Karpathy, Blog on "Unreasonable Effectiveness of Recurrent Neural Networks"

---



---

# Features RNN Captures in C Language?

```
for (i = 0; i < 16; i++) {
    if (k & (1 << i))
        pipe = (in_use & UMXTHREAD_UNCCA) +
            ((count & 0x00000000fffff8) & 0x000000f) << 8;
    if (count == 0)
        sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
}
/* Free our user pages pointer to place camera if all dash */
```



---

# Cell That Activates Inside IF Statements

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

---



---

# Cell That Is Sensitive To Indentation

- Can be interpreted as tracking indentation of code.
- Increasing strength as indentation increases

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

---



---

# Non-Interpretable Cells

- Only 5% of the cells show such interesting properties
- Large portion of the cells are not interpretable by themselves

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
*/
```

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”



---

# Visualizing Hidden State Dynamics

Observe changes in hidden state representation overtime

Tool : LSTMVis

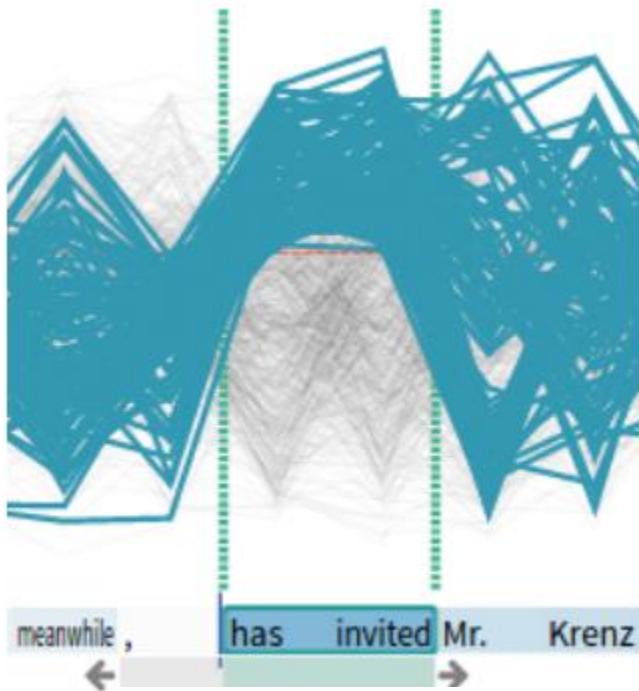
Hendrick et al, "Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks"

---



# Visualizing Hidden State Dynamics

b

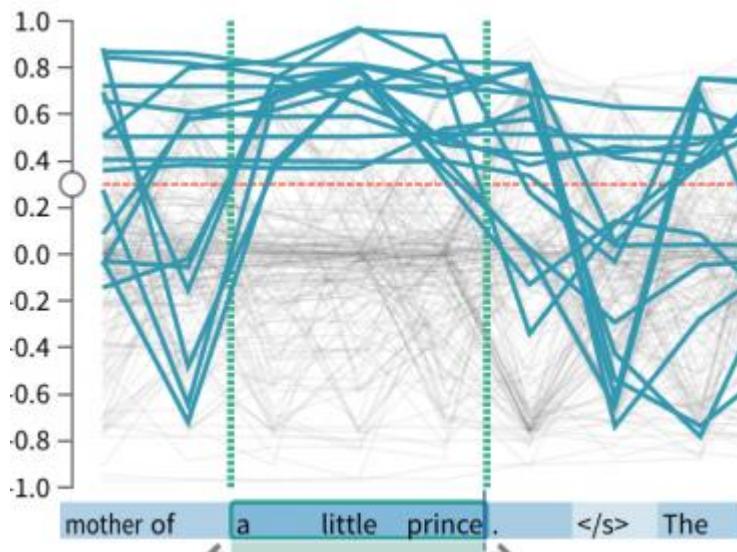


meanwhile ,	has	invited	Mr.	Krenz
however ,	has	n't	been	bad enough
) ,	has	used	his	position to
all ,	has	mainly	been	for the
said ,	has	n't	yet	determined what
Commission ,	has	promised	Poland	and Hungary
however ,	have	n't	stopped	asset-backed securities
central bank	has	allowed	a	key interest
Inc. ,	has	failed	to	make about
Life ,	has	agreed	not	to make
Sen regime	has	sent	thousands of	<unk>
resigned ,	has	helped	renew	calls for
) ,	has	only	one	produced picture

Hendrick et al, "Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks"



# Visualizing Hidden State Dynamics



mother	of	a	little	prince.	</s>
mother	of	a	little	prince.	</s>
wife	in	a	little	hut	,
presence	of	a	little	old	woman.
lived	in	a	little	cottage	with her
her	in	a	great	nobleman	; and
,	in	a	white	coat	and a
not	in	a	good	temper	,
hare	in	a	fishing	net	and fastened

Hendrick et al, "Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks"



---

# Key Takeaways

**Deeper RNNs** are more expressive

- Feedforward depth

- Recurrent depth

**Long term dependencies** are a major problem in RNNs. Solutions:

- Intelligent weight initialization

- LSTMs / GRUs

**Regularization** helps

- Batch Norm: faster convergence

- Dropout: better generalization

**Visualization** helps

- Analyze finer details of features produced by RNNs



---

# References

## Survey Papers

Lipton, Zachary C., John Berkowitz, and Charles Elkan. [A critical review of recurrent neural networks for sequence learning](#), arXiv preprint arXiv:1506.00019 (2015).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. [Chapter 10: Sequence Modeling: Recurrent and Recursive Nets](#). MIT Press, 2016.

## Training

Semeniuta, Stanislau, Aliaksei Severyn, and Erhardt Barth. [Recurrent dropout without memory loss](#). arXiv preprint arXiv:1603.05118 (2016).

Arjovsky, Martin, Amar Shah, and Yoshua Bengio. [Unitary evolution recurrent neural networks](#). arXiv preprint arXiv:1511.06464 (2015).

Le, Quoc V., Navdeep Jaitly, and Geoffrey E. Hinton. [A simple way to initialize recurrent networks of rectified linear units](#). arXiv preprint arXiv:1504.00941 (2015).

Cooijmans, Tim, et al. [Recurrent batch normalization](#). arXiv preprint arXiv:1603.09025 (2016).



---

# References (contd)

## Architectural Complexity Measures

- Zhang, Saizheng, et al. [Architectural Complexity Measures of Recurrent Neural Networks](#). Advances in Neural Information Processing Systems. 2016.
- Pascanu, Razvan, et al. [How to construct deep recurrent neural networks](#). arXiv preprint arXiv:1312.6026 (2013).

## RNN Variants

- Zilly, Julian Georg, et al. [Recurrent highway networks](#). arXiv preprint arXiv:1607.03474 (2016)
- Chung, Junyoung, Sungjin Ahn, and Yoshua Bengio. [Hierarchical multiscale recurrent neural networks](#), arXiv preprint arXiv:1609.01704 (2016).

## Visualization

- Karpathy, Andrej, Justin Johnson, and Li Fei-Fei. [Visualizing and understanding recurrent networks](#). arXiv preprint arXiv:1506.02078 (2015).
- Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, Alexander M. Rush. [LSTMVis: Visual Analysis for RNN](#), arXiv preprint arXiv:1606.07461 (2016).



---

# Appendix



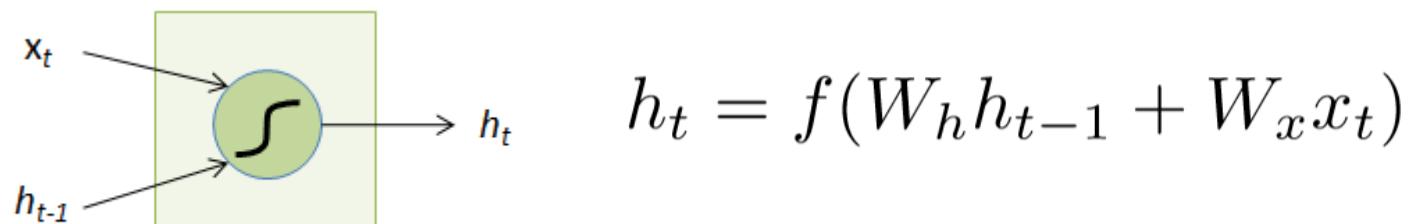
---

# Why go deep?



---

## Another Perspective of the RNN

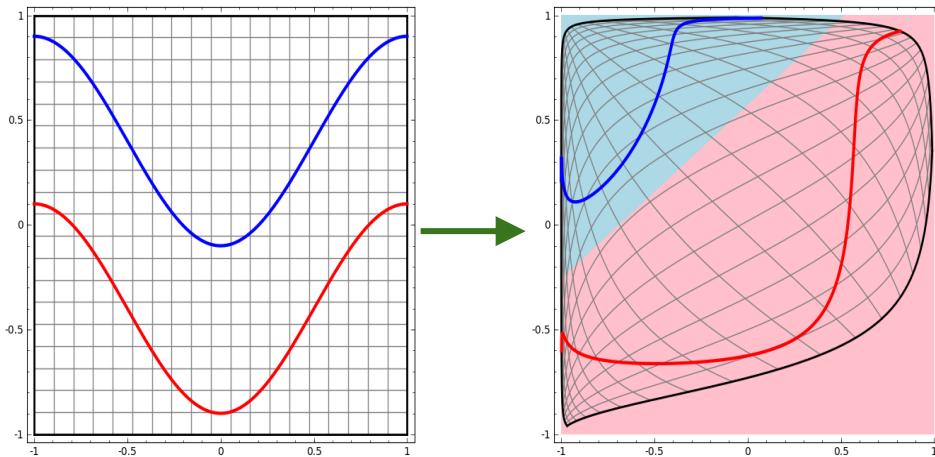


- Affine transformation + element-wise non-linearity
- It is equivalent to one fully connected layer feedforward NN
- Shallow transformation



---

# Visualizing Shallow Transformations



Linear separability is achieved!

The Fully Connected Layer does  
2 things:  
1: Stretch / Rotate (affine)  
2: Distort (non-linearity)

Source: <http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/>

---

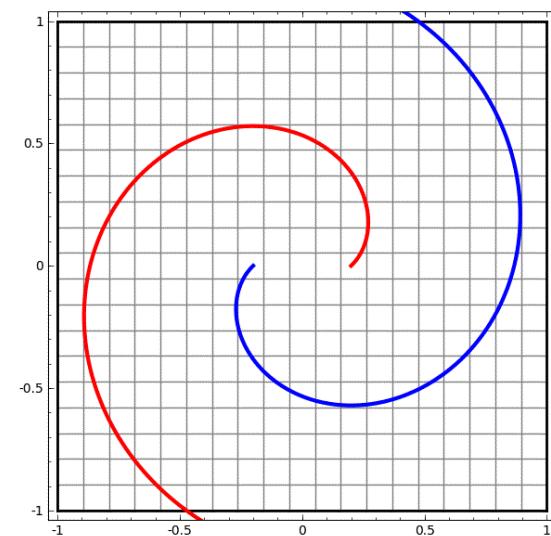


---

# Shallow isn't always enough

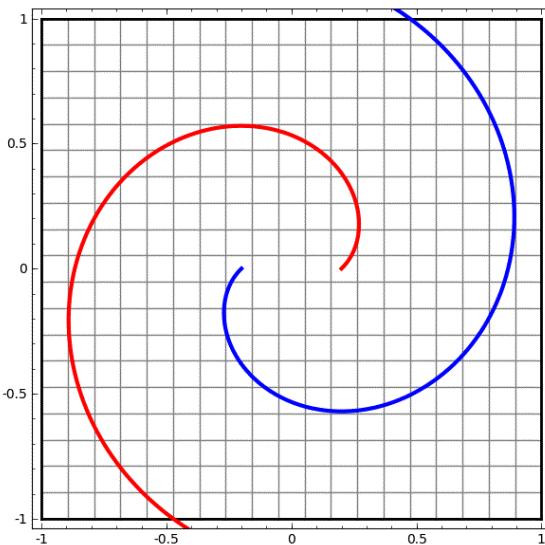
Linear Separability **may not be achieved** for more complex datasets using **just one layer**  $\Rightarrow$  NN isn't expressive enough!

Need more  
layers.

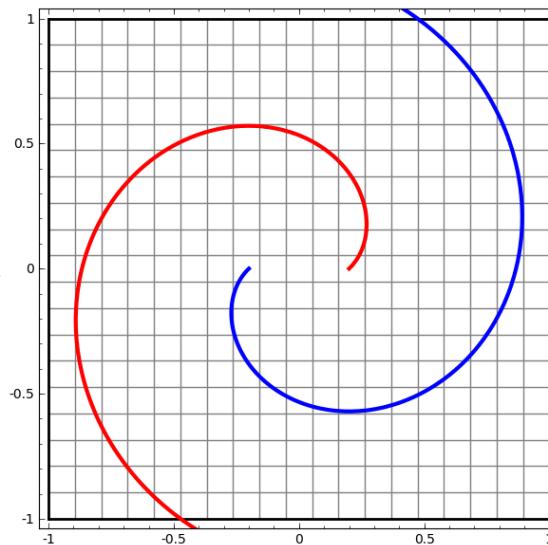




# Visualizing Deep Transformations



4 layers,  
tanh  
activation  
~~Linear~~  
separability!



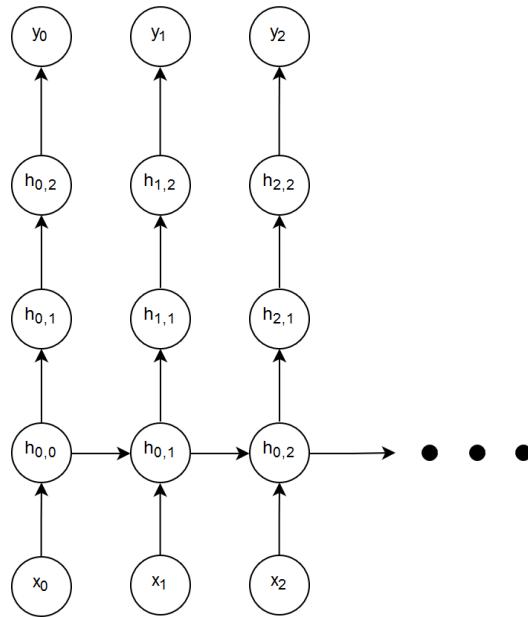
Deeper networks utilize high  
level features  $\Rightarrow$  more expressive!

Can you tell apart the effect of each layer?

Source: <http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/>

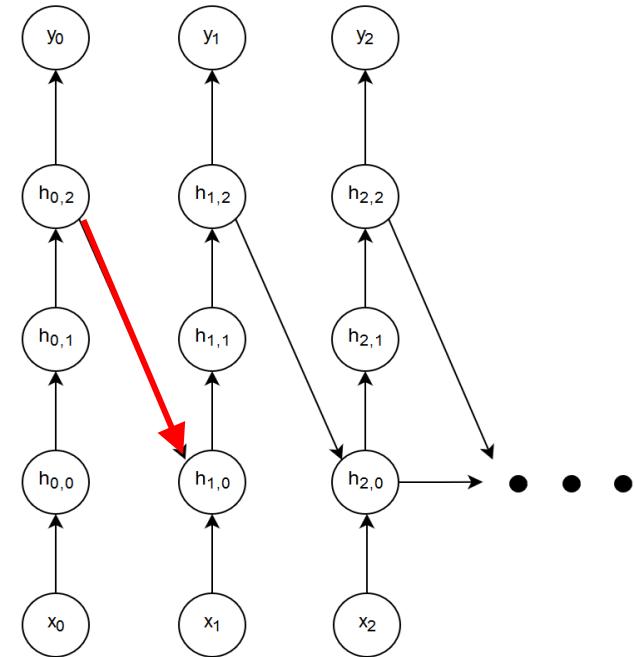


# Which is more expressive?



Recurrent depth = 1  
Feedforward depth = 4

Higher level  
features passed  
on  $\Rightarrow$  win!



Recurrent depth = 3  
Feedforward depth = 4



---

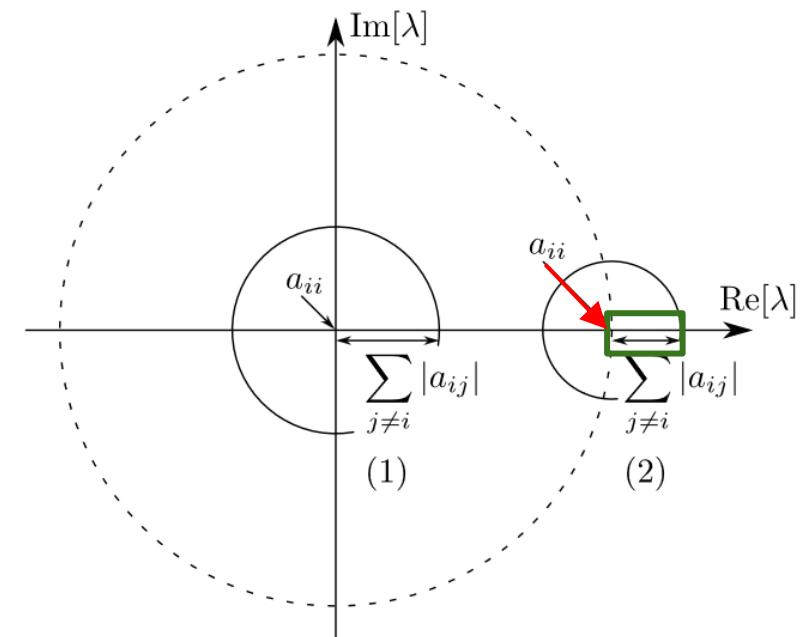
# Gershgorin Circle Theorem (GCT)



# Gershgorin Circle Theorem (GCT)

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & & & \vdots \\ a_{31} & a_{32} & a_{33} & & & \vdots \\ \vdots & & & \ddots & & \vdots \\ \vdots & & & & \ddots & \vdots \\ a_{n1} & \dots & \dots & \dots & \dots & a_{nn} \end{bmatrix}$$

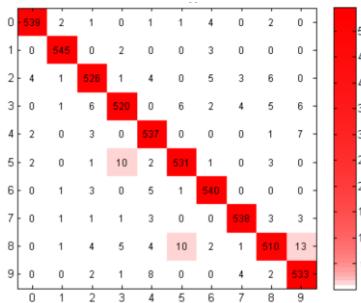
**For any square matrix:** The set of all eigenvalues is the **union of circles** whose **centers** are  $a_{ii}$  and the **radii** are  $\sum_{j \neq i} |a_{ij}|$



Zilly, Julian Georg, et al. "Recurrent highway networks." *arXiv preprint arXiv:1607.03474* (2016).

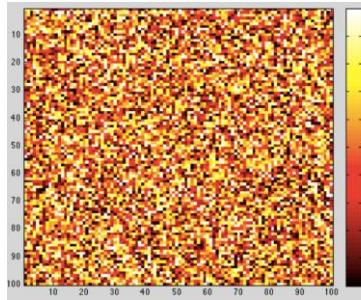


# Implications of GCT



Nearly diagonal matrix

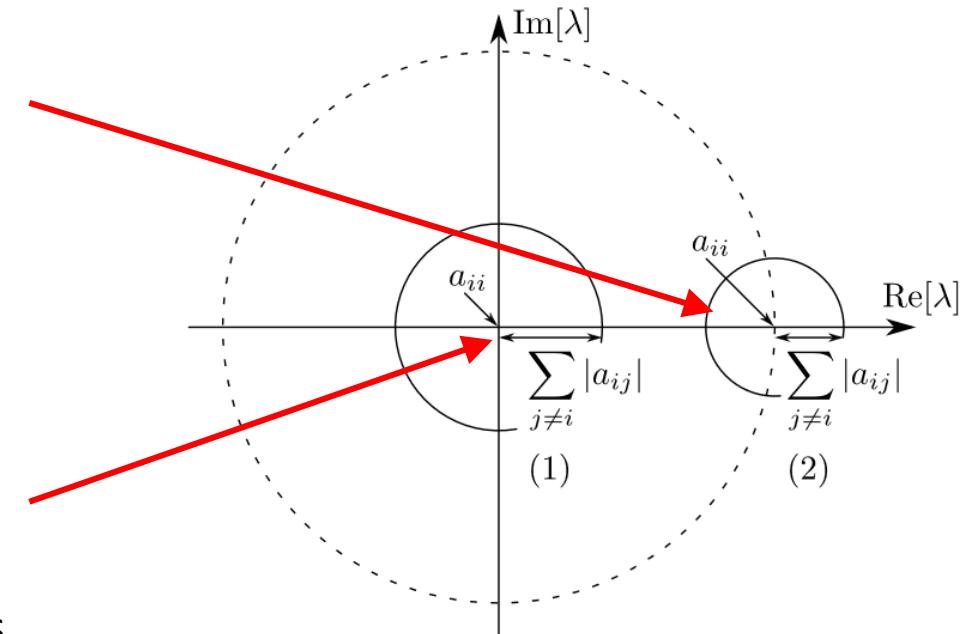
Source: [https://de.mathworks.com/products/demos/machine-learning/handwriting\\_recognition/handwriting\\_recognition.html](https://de.mathworks.com/products/demos/machine-learning/handwriting_recognition/handwriting_recognition.html)



Diffused matrix  
(strong off-diagonal terms),  
mean of all terms  
 $= 0$

Source: <https://i.stack.imgur.com/9inAk.png>

Zilly, Julian Georg, et al. "Recurrent highway networks." *arXiv preprint arXiv:1607.03474* (2016).





---

# More Weight Initialization Methods



---

## Weight Initialization Trick #2: np-RNN

- $W_h$  positive semi-definite (+ve real eigenvalues)
- Activation Function: ReLUs 1, others all less than equal to one

$$\begin{aligned} A &= \frac{1}{N} \langle R^T, R \rangle \\ e &= \max(\lambda(A + I)) \\ W_{hh} &= \frac{I + A}{e} \end{aligned}$$

Geoffrey et al, "Improving Performance of Recurrent Neural Network with ReLU nonlinearity"

- R: standard normal matrix, values drawn from a Gaussian distribution with mean zero and unit variance
- N: size of R
- $\langle , \rangle$  dot product
- e: Maximum eigenvalue of  $(A+I)$



---

## Weight Initialization Trick #3: Unitary Matrix

Unitary Matrix:  $W_h W_h^* = I$  (note: weight matrix is now complex!)

( $W_h^*$  is the complex conjugate matrix of  $W_h$ )

All eigenvalues of  $W_h$  have absolute value 1

---

Arjovsky, Martin, Amar Shah, and Yoshua Bengio. "Unitary evolution recurrent neural networks." 2015).



---

# Challenge: Keeping a Matrix Unitary over time

**Efficient Solution:** Parametrize the matrix

$$\mathbf{W} = \boxed{\mathbf{D}_3 \mathbf{R}_2 \mathcal{F}^{-1} \mathbf{D}_2 \Pi \mathbf{R}_1 \mathcal{F} \mathbf{D}_1}$$

Rank 1 Matrices derived  
from vectors

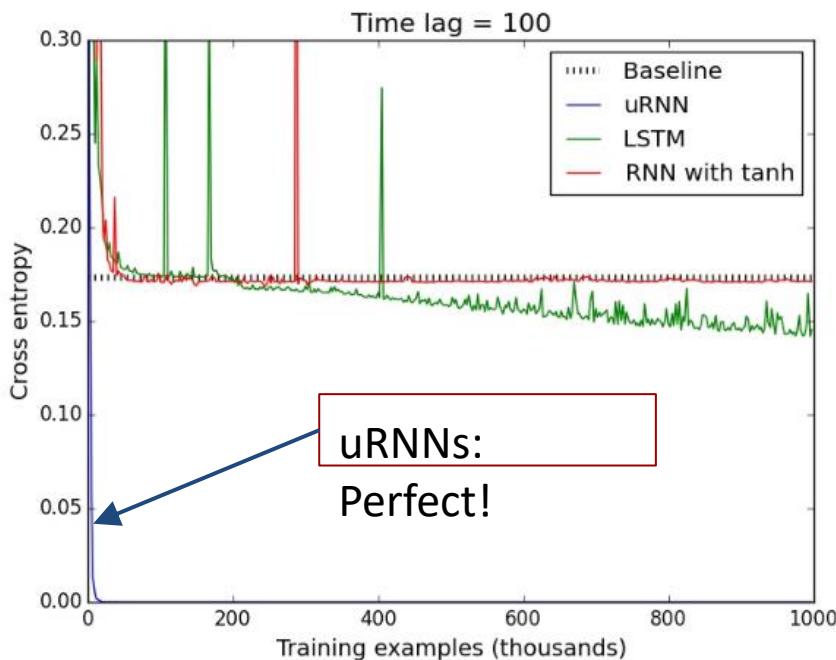
- Storage and updates:  $O(n)$ : efficient!

Arjovsky, Martin, Amar Shah, and Yoshua Bengio. "Unitary evolution recurrent neural networks." 2015).

---



# Results for the Copying Memory Problem



- Input:

$a_1 a_2 \dots a_{10} 0 0 0 0 0 \dots 0$   
10 T zeros  
symbols

- Output:  $a_1 \dots a_{10}$

- **Challenge:** Remembering symbols over an arbitrarily large time gap

Cross entropy for the copying memory

Arjovsky, Marten, Amar Shah, and Yoshua Bengio. "Unitary evolution recurrent neural networks." 2015).



# Summary

Model	I-RNN	np-RNN	Unitary-RNN
Activation Function	ReLU	ReLU	ReLU
Initialization	Identity Matrix	Positive Semi-definite (normalized eigenvalues)	Unitary Matrix
Performance compared to LSTM	Less than or equal	Equal	Greater
Benchmark Tasks	Action Recognition, Addition, MNIST	Action Recognition, Addition MNIST	Copying Problem, Adding Problem
Sensitivity to	High	Low	Low



---

# Dropout



$$c_t = d(f_t * c_{t-1} + i_t * g_t)$$

## Model Moon (2015)

Able to learn long term dependencies, not capable of exploiting them during test phase

Test time equations for GRU,

$$\mathbf{h}_t = (\mathbf{h}_{t-1} + \mathbf{g}_t)p$$

$$\mathbf{h}_t = ((\mathbf{h}_{t-2} + \mathbf{g}_{t-1})p + \mathbf{g}_t)p$$

$$\mathbf{h}_t = p^{t+1}\mathbf{h}_0 + \sum_{i=0}^t p^{t-i+1}\mathbf{g}_i$$

- P is the probability to not drop a neuron
- For large t, hidden state contribution is close to zero during test



# Model Barth (2016)

$$c_t = f_t * c_{t-1} + i_t * d(g_t)$$

Drop differences that are added to the network, not the actual values

Barth (2016)

Allows to use per-step dropout

Test time equation after recursion,

$$\mathbf{h}_t = p\mathbf{h}_0 + \sum_{i=0}^t p \mathbf{g}_i = p\mathbf{h}_0 + p \sum_{i=0}^t \mathbf{g}_i$$

- P is the probability to not drop a neuron
- For large t, hidden state contribution is retained as at train time



---

# Visualization

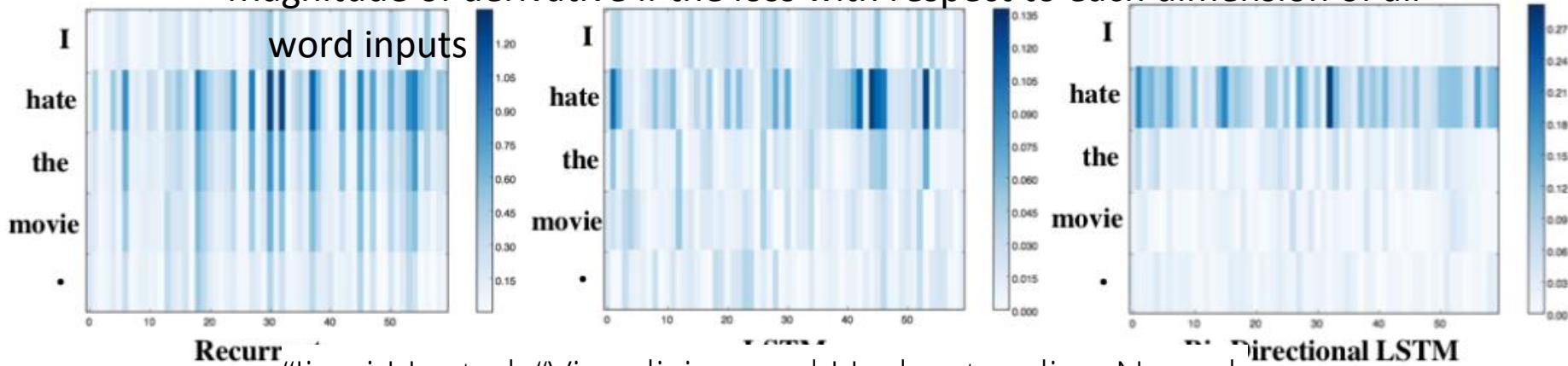


# Visualize gradients: Saliency maps

Categorize phrase/sentence into (v.positive, positive, neutral, negative, v.negative)

How much each unit contributes to the decision ?

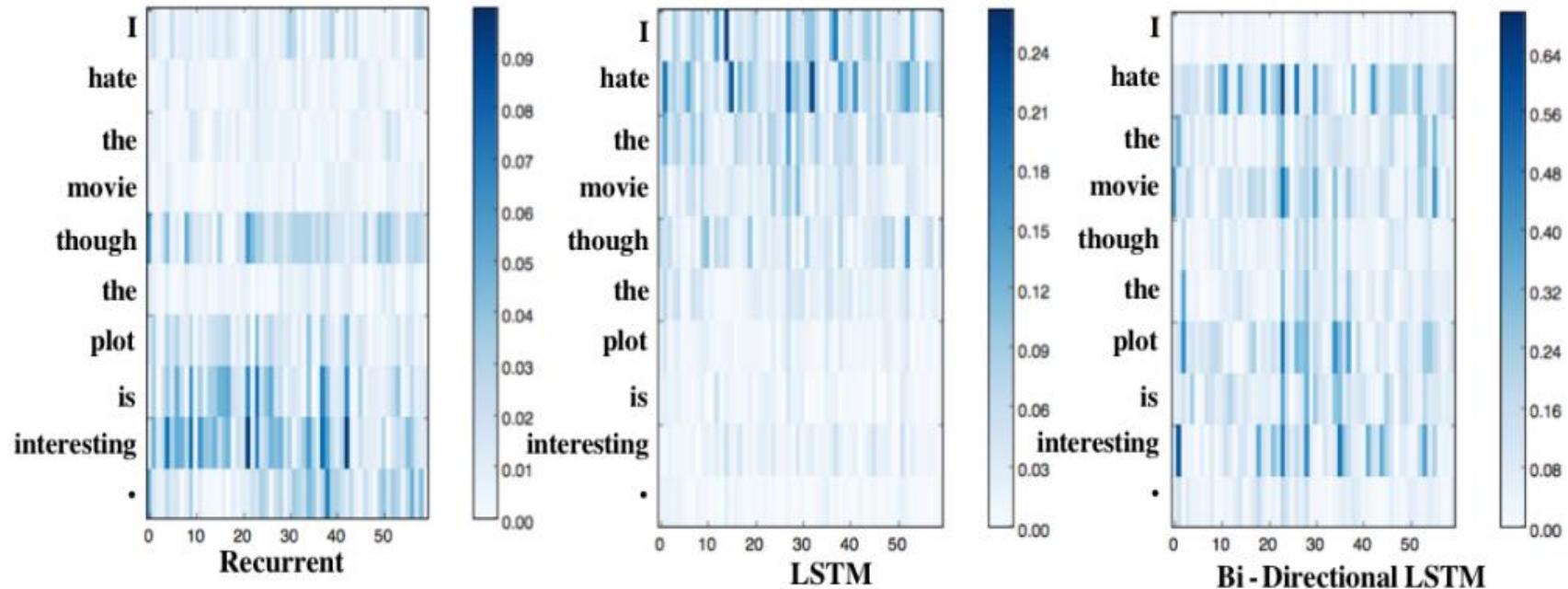
Magnitude of derivative if the loss with respect to each dimension of all



"Jiwei LI et al, "Visualizing and Understanding Neural Models in NLP"



# Visualize gradients: Saliency maps



"Jiwei LI et al, "Visualizing and Understanding Neural Models in NLP"



---

# Error Analysis

N-gram Errors

Dynamic n-long memory Errors

Rare word Errors

Word model Errors

Punctuation Errors

Boost Errors

“Karpathy et al, Visualizing and Understanding  
Recurrent Networks”

---



## 50K -> 500K parameter model

50K -> 500K parameter model	
Reduced Total Errors	44K (184K-140K)
N-gram Error	81% (36K/44K)
Dynamic n-long memory Errors	1.7% (0.75K/44k)
Rare words Error	1.7% (0.75K/44K)
Word model Error	1.7% (0.75K/44k)
Punctuation Error	1.7% (0.75K/44K)
Boost Error	11.36% (5K/44K)



---

# Error Analysis: Conclusions

- N-gram Errors
  - Scale model
- Dynamic n-long memory
  - Memory Networks
- Rare words
  - Increase training size
- Word Level Predictions/ Punctuations:
  - Hierarchical context models
    - Stacked Models
    - GF RNN, CW RNN

"Karpathy et al, Visualizing and Understanding Recurrent Networks"

---



---

# Recurrent Highway Networks



---

# Understanding Long Term Dependencies from Jacobian

Learning long term dependencies is a challenge because:

If the Jacobian has a spectral radius (absolute largest eigenvalue)  $< 1$ , the network faces **vanishing gradients**. Here it happens if  $\gamma\sigma_{max} < 1$

Hence, ReLU's are an attractive option! They have  $\sigma_{max} = 1$  (given at least one positive element)

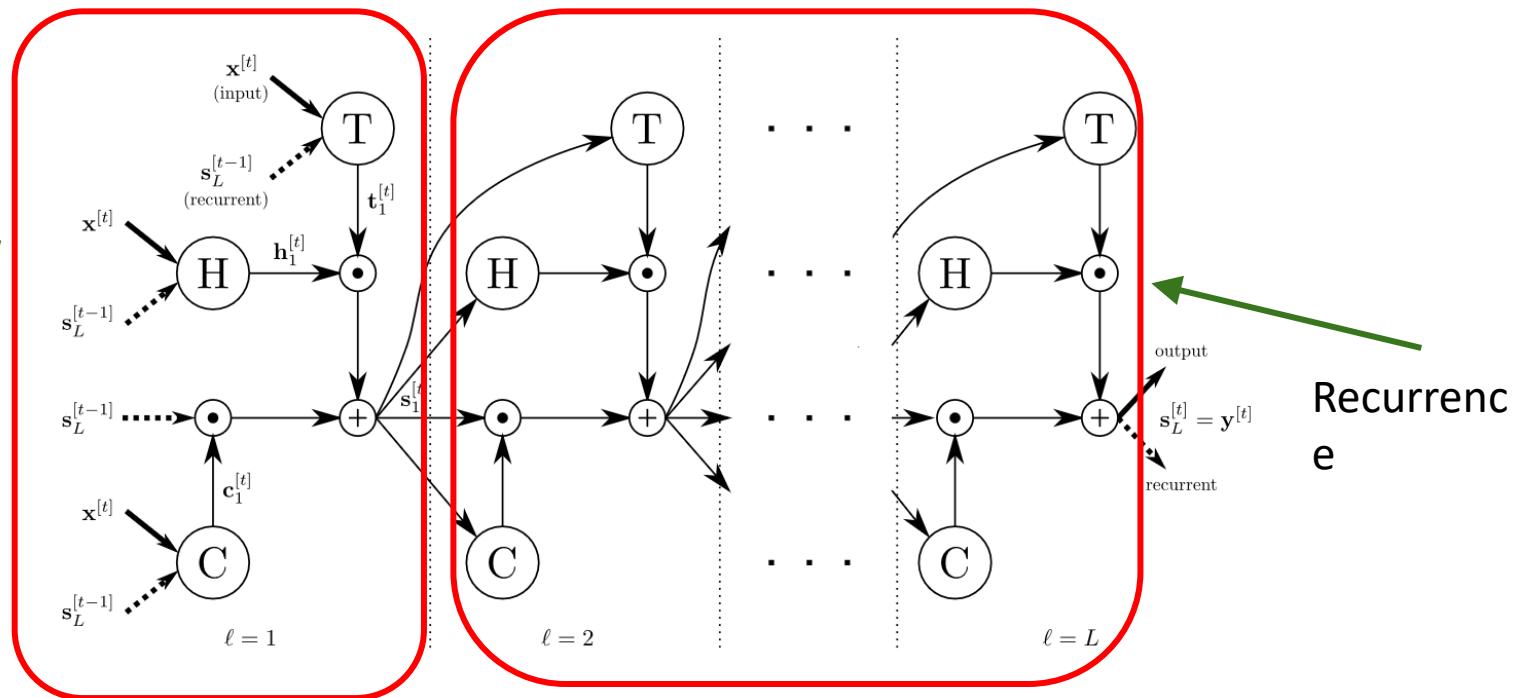
If the Jacobian has a spectral radius  $> 1$ , the network faces **exploding gradients**

---



# Recurrent Highway Networks (RHN)

LSTM  
!



Zilly, Julian Georg, et al. "Recurrent highway networks." *arXiv preprint arXiv:1607.03474* (2016).



# RHN Equations

RHN:

Recurrent depth

Feedforward depth (not shown)

Input transformations:

T, C: Transform, Carry operators

RHN Output:

State update equation:  $y = \mathbf{h} \cdot \mathbf{t} + \mathbf{x} \cdot \mathbf{c}$  Note: h is transformed input, y is state

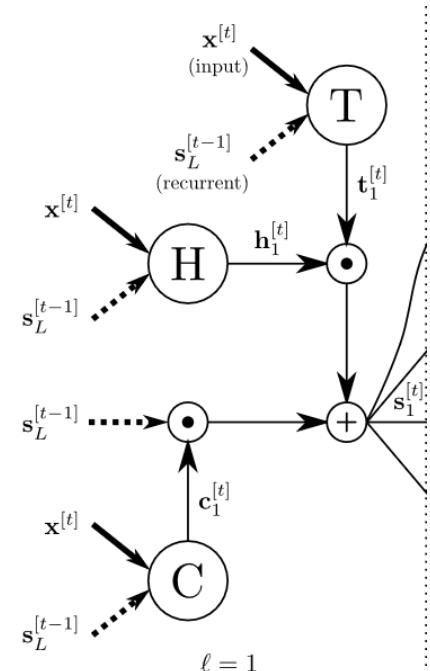
$$\begin{aligned}\mathbf{h} &= H(x, \mathbf{W}_H) \\ \mathbf{t} &= T(x, \mathbf{W}_T) \\ \mathbf{c} &= C(x, \mathbf{W}_C)\end{aligned}$$

$$\begin{aligned}s_\ell^{[t]} &= \mathbf{h}_\ell^{[t]} \cdot \mathbf{t}_\ell^{[t]} + s_{\ell-1}^{[t]} \cdot \mathbf{c}_\ell^{[t]} \\ \mathbf{h}_\ell^{[t]} &= \tanh(\mathbf{W}_H \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{H\ell} s_{\ell-1}^{[t]} + \mathbf{b}_{H\ell}), \\ \mathbf{t}_\ell^{[t]} &= \sigma(\mathbf{W}_T \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{T\ell} s_{\ell-1}^{[t]} + \mathbf{b}_{T\ell}), \\ \mathbf{c}_\ell^{[t]} &= \sigma(\mathbf{W}_C \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{C\ell} s_{\ell-1}^{[t]} + \mathbf{b}_{C\ell}),\end{aligned}$$

Indicator  
function

Recurrence layer  
Index

Zilly, Julian Georg, et al. "Recurrent highway networks." *arXiv preprint arXiv:1607.03474* (2016).





---

# Gradient Equations in RHN

$$\mathbf{y}^{[t]} = \mathbf{h}^{[t]} \cdot \mathbf{t}^{[t]} + \mathbf{y}^{[t-1]} \cdot \mathbf{c}^{[t]}$$

For an RHN with **recurrence depth 1**, RHN Output is:

Jacobian is simple:

But the gradient of  $\mathbf{A}$  is  $\frac{\partial \mathbf{y}^{[t]}}{\partial \mathbf{y}^{[t-1]}}$   $\mathbf{A} = \text{diag}(\mathbf{c}^{[t]}) + \mathbf{H}' \text{diag}(\mathbf{t}^{[t]}) + \mathbf{C}' \text{diag}(\mathbf{y}^{[t-1]}) + \mathbf{T}' \text{diag}(\mathbf{h}^{[t]})$

where  $\mathbf{H}' = \mathbf{R}_H^\top \text{diag}[\tanh'(\mathbf{R}_H \mathbf{y}^{[t-1]})]$ ,

$$\mathbf{T}' = \mathbf{R}_T^\top \text{diag}[\sigma'(\mathbf{R}_T \mathbf{y}^{[t-1]})],$$

Using  $\mathbf{C}' = \mathbf{R}_C^\top \text{diag}[\sigma'(\mathbf{R}_C \mathbf{y}^{[t-1]})]$ , of the circles are:

The radius are.

$$\mathbf{c}_i^{[t]} + \mathbf{H}'_{ii} \mathbf{t}_i^{[t]} + \mathbf{C}'_{ii} \mathbf{y}_i^{[t-1]} + \mathbf{T}'_{ii} \mathbf{h}_i^{[t]}$$

$$\sum_{j=1, j \neq i}^n |\mathbf{H}'_{ij} \mathbf{t}_i^{[t]} + \mathbf{C}'_{ij} \mathbf{y}_i^{[t-1]} + \mathbf{T}'_{ij} \mathbf{h}_i^{[t]}|$$

The eigenvalues lie within these circles



---

# Analysis

$$\text{Centers: } \mathbf{c}_i^{[t]} + \mathbf{H}'_{ii} \mathbf{t}_i^{[t]} + \mathbf{C}'_{ii} \mathbf{y}_i^{[t-1]} + \mathbf{T}'_{ii} \mathbf{h}_i^{[t]}, \text{ radii: } \sum_{j=1, j \neq i}^n |\mathbf{H}'_{ij} \mathbf{t}_i^{[t]} + \mathbf{C}'_{ij} \mathbf{y}_i^{[t-1]} + \mathbf{T}'_{ij} \mathbf{h}_i^{[t]}|$$

If we wish to completely remember the previous state:  $\mathbf{c} = \mathbf{1}, \mathbf{t} = \mathbf{0}$

Saturation  $\Rightarrow \mathbf{T}' = \mathbf{C}' = \mathbf{0}_{nxn}$

Thus, centers ( $\lambda$ ) are  $\mathbf{1}$ , radii are  $\mathbf{0}$

If we wish to completely forget the previous state:  $\mathbf{c} = \mathbf{0}, \mathbf{t} = \mathbf{1}$

Eigenvalues are those of  $\mathbf{H}'$

Possible to span the spectrum between these two cases by adjusting the Jacobian  $\mathbf{A}$

(\*) Increasing depth improves expressivity

---



# Results

Model	Size	Best Val.	Test
RNN-LDA + KN-5 + cache (Mikolov & Zweig, 2012)	9 M	—	92.0
Conv.+Highway+LSTM+dropout (Kim et al., 2015)	19 M	—	78.9
LSTM+dropout (Zaremba et al., 2014)	66 M	82.2	78.4
Variational LSTM (Gal, 2015)	66 M	77.3	75.0
Variational LSTM + WT (Press & Wolf, 2016)	51 M	75.8	73.2
Pointer Sentinel-LSTM (Merity et al., 2016)	21 M	72.4	70.9
Variational LSTM + WT + augmented loss (Inan et al., 2016)	51 M	71.1	68.5
<b>Variational RHN</b>	<b>32 M</b>	<b>71.2</b>	<b>68.5</b>
Neural Architecture Search with base 8 (Zoph & Le, 2016)	32 M	—	67.9
<b>Variational RHN + WT</b>	<b>23 M</b>	<b>67.9</b>	<b>65.4</b>
Neural Architecture Search with base 8 + WT (Zoph & Le, 2016)	25 M	—	64.0
Neural Architecture Search with base 8 + WT (Zoph & Le, 2016)	54 M	—	62.4

BPC on Penn Treebank

Model	BPC	Size
Grid-LSTM (Kalchbrenner et al., 2015)	1.47	17 M
MI-LSTM (Wu et al., 2016)	1.44	17 M
mlSTM (Krause et al., 2016)	1.42	21 M
LN HyperNetworks (Ha et al., 2016)	1.34	27 M
LN HM-LSTM (Chung et al., 2016)	1.32	35 M
<b>RHN - Rec. depth 5</b>	<b>1.31</b>	<b>23 M</b>
<b>RHN - Rec. depth 10</b>	<b>1.30</b>	<b>21 M</b>
<b>Large RHN - Rec. depth 10</b>	<b>1.27</b>	<b>46 M</b>

BPC on enwiki8 (Hutter Prize)

Model	BPC	Size
MI-LSTM (Wu et al., 2016)	1.44	17 M
mLSTM (Krause et al., 2016)	1.40	10 M
BN LSTM (Cooijmans et al., 2016)	1.36	16 M
HM-LSTM (Chung et al., 2016)	1.32	35 M
LN HM-LSTM (Chung et al., 2016)	1.29	35 M
<b>RHN - Rec. depth 10</b>	<b>1.29</b>	<b>20 M</b>
<b>Large RHN - Rec. depth 10</b>	<b>1.27</b>	<b>45 M</b>

BPC on text8 (Hutter Prize)



---

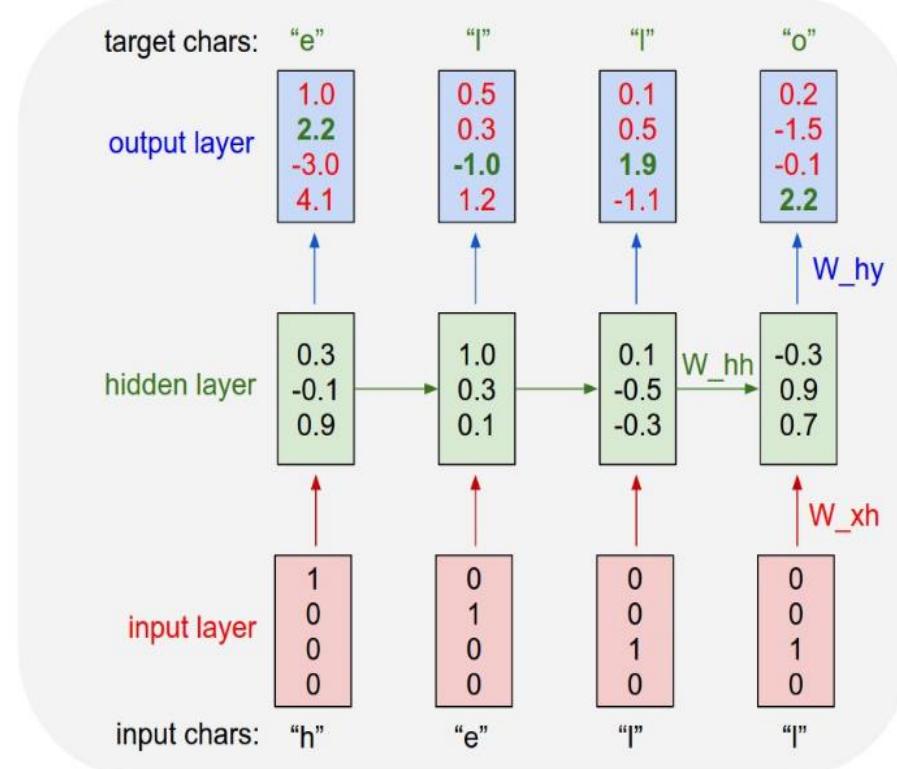
# LSTMs for Language Models



# LSTMs are Very Effective!

## Application: Language Model

Task: Predicting the next character given the current character





---

## Train Input: Wikipedia Data

Hutter Prize 100 MB Dataset of raw wikipedia, 96 MB for training  
Trained overnight on a LSTM



## Generated Text:

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servitious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (P?)<http://www.humah.yahoo.com/guardian.cfm/7754800786d1/551963s89.htm> Official economics Adjoint for the Nazism, Montgomery was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]

- Remembers to close a bracket
- Capitalize nouns
- 404 Page Not Found! :P The LSTM hallucinates it.



---

## Train Input:

16MB of Latex source of algebraic stacks/geometry

Trained on Multi-Layer LSTM

## Test Output

Generated Latex files “almost” compile, the authors had to fix some issues manually

We will look at some of these errors

---



# Generated Latex Source Code

```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on $\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \rightarrow \mathcal{I}$
is an injective and let $\mathfrak{q}$ be an abelian sheaf on $X$.
Let $\mathcal{F}$ be a fibered complex. Let $\mathcal{F}$ be a category.

\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let $\mathcal{F}$ be an abelian quasi-coherent sheaf on $\mathcal{C}$.
Let $\mathcal{F}$ be a coherent $\mathcal{O}_X$-module. Then
$\mathcal{F}$ is an abelian catenary over $\mathcal{C}$.
\item The following are equivalent
\begin{enumerate}
\item $\mathcal{F}$ is an $\mathcal{O}_X$-module.
\end{enumerate}
\end{enumerate}
\end{proof}
```

- Begins with a proof but ends with a lemma
- Begins enumerate but does not end it
- Likely because of the long term dependency.
- Can be reduced with larger/better models



# Compiled Latex Files: Hallucinated Algebra

For  $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m,n} = 0$ , hence we can find a closed subset  $\mathcal{H}$  in  $\mathcal{H}$  and any sets  $\mathcal{F}$  on  $X$ ,  $U$  is a closed immersion of  $S$ , then  $U \rightarrow T$  is a separated algebraic space.

*Proof.* Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by  $\coprod Z \times_U U \rightarrow V$ . Consider the maps  $M$  along the set of points  $\text{Sch}_{fppf}$  and  $U \rightarrow U$  is the fibre category of  $S$  in  $U$  in Section ?? and the fact that any  $U$  affine, see Morphisms, Lemma ???. Hence we obtain a scheme  $S$  and any open subset  $W \subset U$  in  $\text{Sh}(G)$  such that  $\text{Spec}(R') \rightarrow S$  is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that  $f_i$  is of finite presentation over  $S$ . We claim that  $\mathcal{O}_{X,x}$  is a scheme where  $x, x', s'' \in S'$  such that  $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}'_{X',x'}$  is separated. By Algebra, Lemma ?? we can define a map of complexes  $\text{GL}_{S'}(x'/S'')$  and we win.  $\square$

To prove study we see that  $\mathcal{F}|_U$  is a covering of  $\mathcal{X}'$ , and  $\mathcal{T}_i$  is an object of  $\mathcal{F}_{X/S}$  for  $i > 0$  and  $\mathcal{F}_p$  exists and let  $\mathcal{F}_i$  be a presheaf of  $\mathcal{O}_X$ -modules on  $\mathcal{C}$  as a  $\mathcal{F}$ -module. In particular  $\mathcal{F} = U/\mathcal{F}$  we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)^{\text{opp}}_{fppf}, (\text{Sch}/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \hookrightarrow (U, \text{Spec}(A))$$

is an open subset of  $X$ . Thus  $U$  is affine. This is a continuous map of  $X$  is the inverse, the groupoid scheme  $S$ .

*Proof.* See discussion of sheaves of sets.  $\square$

The result for prove any open covering follows from the less of Example ???. It may replace  $S$  by  $X_{\text{spaces},\text{etale}}$  which gives an open subspace of  $X$  and  $T$  equal to  $S_{\text{Zar}}$ , see Descent, Lemma ???. Namely, by Lemma ?? we see that  $R$  is geometrically regular over  $S$ .

**Lemma 0.1.** Assume (3) and (3) by the construction in the description.

Suppose  $X = \lim |X|$  (by the formal open covering  $X$  and a single map  $\text{Proj}_X(\mathcal{A}) = \text{Spec}(B)$  over  $U$  compatible with the complex

$$\text{Set}(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that  $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$  is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If  $T$  is surjective we may assume that  $T$  is connected with residue fields of  $S$ . Moreover there exists a closed subspace  $Z \subset X$  of  $X$  where  $U$  in  $X'$  is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1)  $f$  is locally of finite type. Since  $S = \text{Spec}(R)$  and  $Y = \text{Spec}(R)$ .

*Proof.* This is form all sheaves of sheaves on  $X$ . But given a scheme  $U$  and a surjective, fppf morphism  $U \rightarrow X$ . Let  $U \cap U = \coprod_{i=1,\dots,n} U_i$  be the scheme  $X$  over  $S$  at the schemes  $X_i \rightarrow X$  and  $U = \lim_i X_i$ .  $\square$

The following lemma surjective restrocomposes of this implies that  $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\dots,x_0}$ .

**Lemma 0.2.** Let  $X$  be a locally Noetherian scheme over  $S$ ,  $E = \mathcal{F}_{X/S}$ . Set  $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$ . Since  $\mathcal{I}^n \subset \mathcal{I}^n$  are nonzero over  $i_0 \leq p$  is a subset of  $\mathcal{J}_{n,0} \circ \mathcal{A}_2$  works.

**Lemma 0.3.** In Situation ???. Hence we may assume  $q' = 0$ .

*Proof.* We will use the property we see that  $p$  is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) \cong \mathcal{O}_{X',\mathcal{O}}$$

where  $K$  is an  $F$ -algebra where  $\delta_{n+1}$  is a scheme over  $S$ .  $\square$

• Generates Lemmas and their proofs

- Equations with correct latex structure
- No, they dont mean anything yet !



# Compiled Latex Files: Hallucinated Algebra

Nice try on the  
diagrams !

*Proof.* Omitted.

**Lemma 0.1.** Let  $\mathcal{C}$  be a set of the construction.

Let  $\mathcal{C}$  be a gerber covering. Let  $\mathcal{F}$  be a quasi-coherent sheaves of  $\mathcal{O}$ -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

*Proof.* This is an algebraic space with the composition of sheaves  $\mathcal{F}$  on  $X_{\text{etale}}$  we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where  $\mathcal{G}$  defines an isomorphism  $\mathcal{F} \rightarrow \mathcal{F}$  of  $\mathcal{O}$ -modules.  $\square$

**Lemma 0.2.** This is an integer  $\mathbb{Z}$  is injective.

*Proof.* See Spaces, Lemma ??.

**Lemma 0.3.** Let  $S$  be a scheme. Let  $X$  be a scheme and  $X$  is an affine open covering. Let  $\mathcal{U} \subset X$  be a canonical and locally of finite type. Let  $X$  be a scheme. Let  $X$  be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let  $X$  be a scheme. Let  $X$  be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

be a morphism of algebraic spaces over  $S$  and  $Y$ .

*Proof.* Let  $X$  be a nonzero scheme of  $X$ . Let  $X$  be an algebraic space. Let  $\mathcal{F}$  be a quasi-coherent sheaf of  $\mathcal{O}_X$ -modules. The following are equivalent

- (1)  $\mathcal{F}$  is an algebraic space over  $S$ .
- (2) If  $X$  is an affine open covering.

Consider a common structure on  $X$  and  $X$  the functor  $\mathcal{O}_X(U)$  which is locally of finite type.  $\square$

This since  $\mathcal{F} \in \mathcal{F}$  and  $x \in \mathcal{G}$  the diagram

$$\begin{array}{ccccc}
 S & \longrightarrow & & & \\
 \downarrow & & & & \\
 \xi & \longrightarrow & \mathcal{O}_{X'} & \nearrow & \\
 \text{gor}_s & & \uparrow & & \\
 & & = a' & \longrightarrow & \\
 & & \downarrow & & \\
 & & = a' & \longrightarrow & \alpha \\
 & & & & \downarrow \\
 \text{Spec}(K_\psi) & & \text{Mor}_{\text{Sets}} & & \text{d}(\mathcal{O}_{X_{/\psi}}, \mathcal{G})
 \end{array}$$

is a limit. Then  $\mathcal{G}$  is a finite type and assume  $S$  is a flat and  $\mathcal{F}$  and  $\mathcal{G}$  is a finite type  $f_*$ . This is of finite type diagrams, and

- the composition of  $\mathcal{G}$  is a regular sequence,
- $\mathcal{O}_{X'}$  is a sheaf of rings.

$\square$

*Proof.* We have see that  $X = \text{Spec}(R)$  and  $\mathcal{F}$  is a finite type representable by algebraic space. The property  $\mathcal{F}$  is a finite morphism of algebraic stacks. Then the cohomology of  $X$  is an open neighbourhood of  $U$ .  $\square$

*Proof.* This is clear that  $\mathcal{G}$  is a finite presentation, see Lemmas ??.

A reduced above we conclude that  $U$  is an open covering of  $C$ . The functor  $\mathcal{F}$  is a “field”

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_x \dashv (\mathcal{O}_{X_{\text{etale}}}) \rightarrow \mathcal{O}_{X_x}^{-1} \mathcal{O}_{X_x}(\mathcal{O}_{X_x}^B)$$

is an isomorphism of covering of  $\mathcal{O}_{X_x}$ . If  $\mathcal{F}$  is the unique element of  $\mathcal{F}$  such that  $X$  is an isomorphism.

The property  $\mathcal{F}$  is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme  $\mathcal{O}_X$ -algebra with  $\mathcal{F}$  are opens of finite type over  $S$ .

If  $\mathcal{F}$  is a scheme theoretic image points.  $\square$

If  $\mathcal{F}$  is a finite direct sum  $\mathcal{O}_{X_x}$  is a closed immersion, see Lemma ??, This is a sequence of  $\mathcal{F}$  is a similar morphism.



# Generative Adversarial Networks

---



# What are GANs?

---

System of two neural networks competing against each other in a zero-sum game framework.

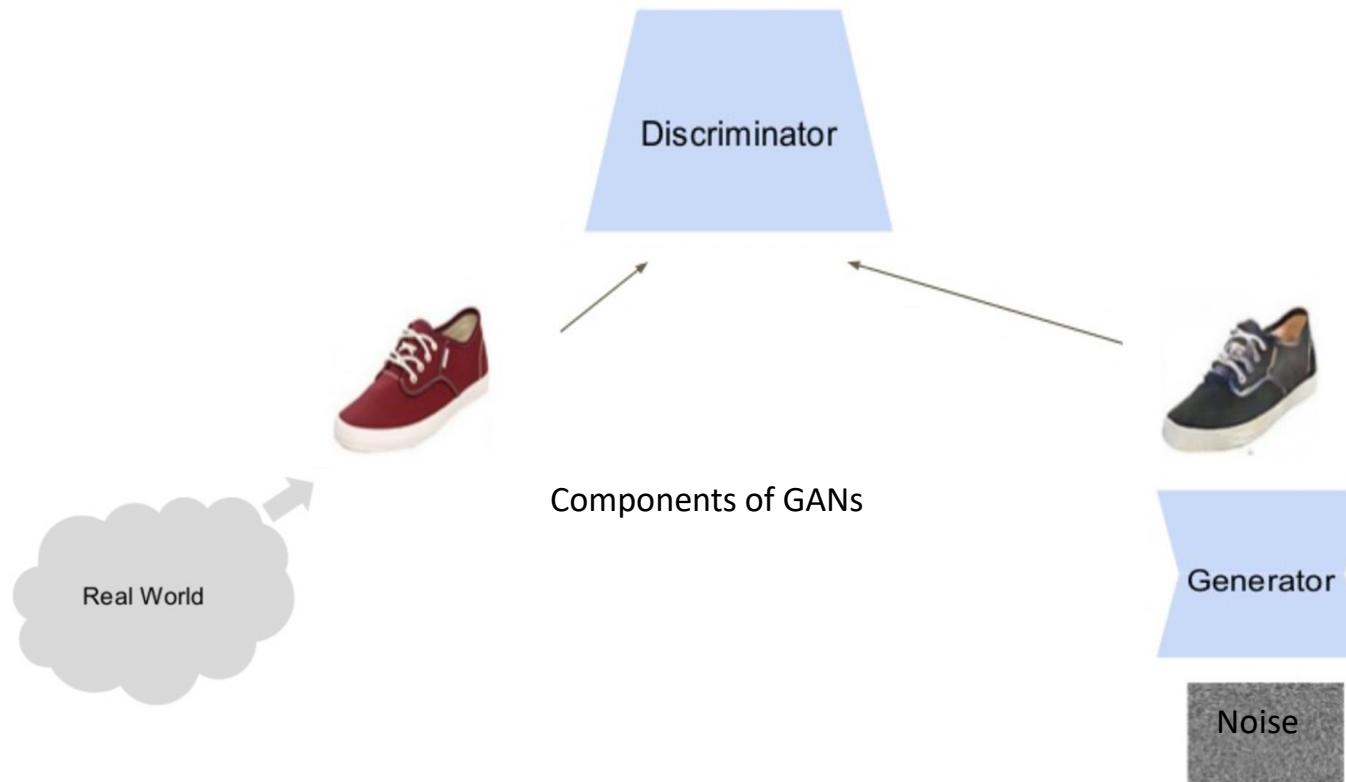
They were first introduced by [Ian Goodfellow](#) *et al.* in 2014.

Can learn to draw samples from a model that is similar to data that we give them.

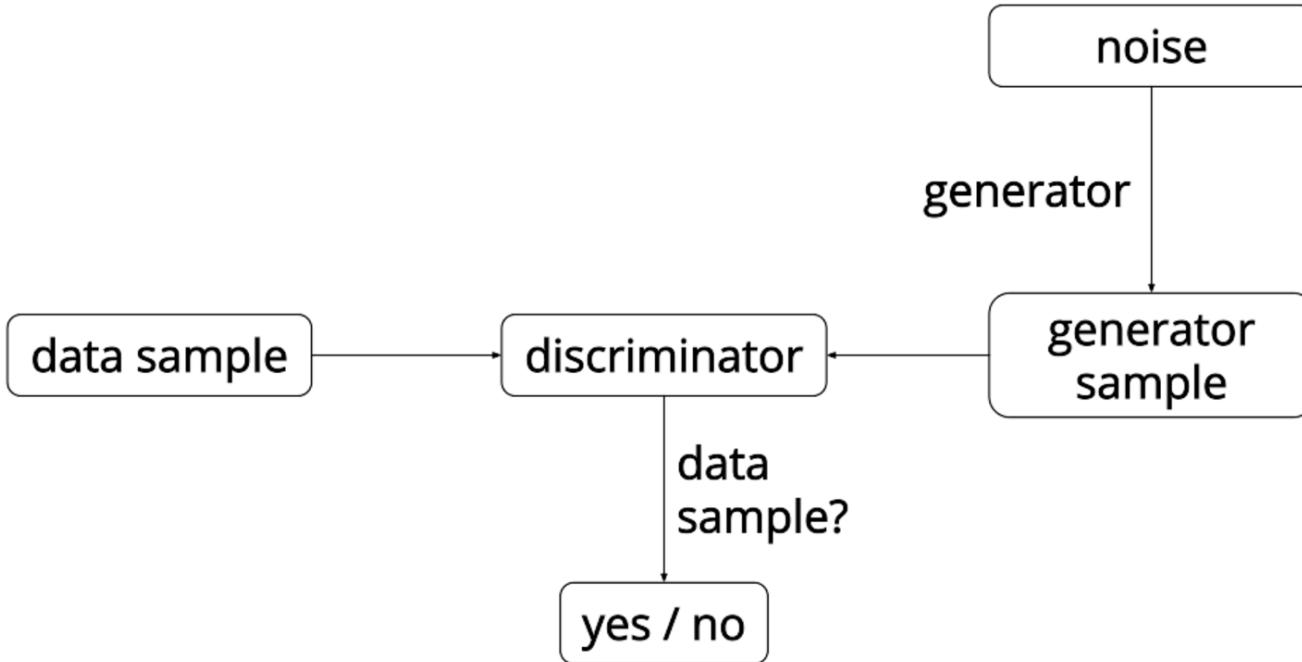
---



True/False



Slide credit – Victor Garcia



*Overview of GANs*

Source: <https://ishmaelbelghazi.github.io/ALI>



# Discriminative Models

---

A **discriminative** model learns a function that maps the input data ( $x$ ) to some desired output class label ( $y$ ).

In probabilistic terms, they directly learn the conditional distribution  $P(y/x)$ .





# Generative Models

---

A **generative** model tries to learn the joint probability of the input data and labels simultaneously i.e.  $P(x,y)$ .

Potential to understand and explain the underlying structure of the input data even when there are no labels.

---



# How GANs are being used?

---

Applied for modelling natural images.

Performance is fairly good in comparison to other generative models.

Useful for unsupervised learning tasks.

---



# Why GANs?

---

Use a latent code.

Asymptotically consistent (unlike variational methods) .

No Markov chains needed.

Often regarded as producing the best samples.

---



man  
with glasses



man  
without glasses



woman  
without glasses



woman with glasses



# How to train GANs?

---

Objective of generative network - increase the error rate of the discriminative network.

Objective of discriminative network – decrease binary classification loss.

Discriminator training - backprop from a binary classification loss.

Generator training - backprop the negation of the binary classification loss of the discriminator.

---



---

## Loss Functions

$$\mathcal{L}(\hat{x}) = \min_{x \in data} (x - \hat{x})^2$$

Generator

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Discriminator

---



Generated bedrooms. Source: “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks” <https://arxiv.org/abs/1511.06434v2>



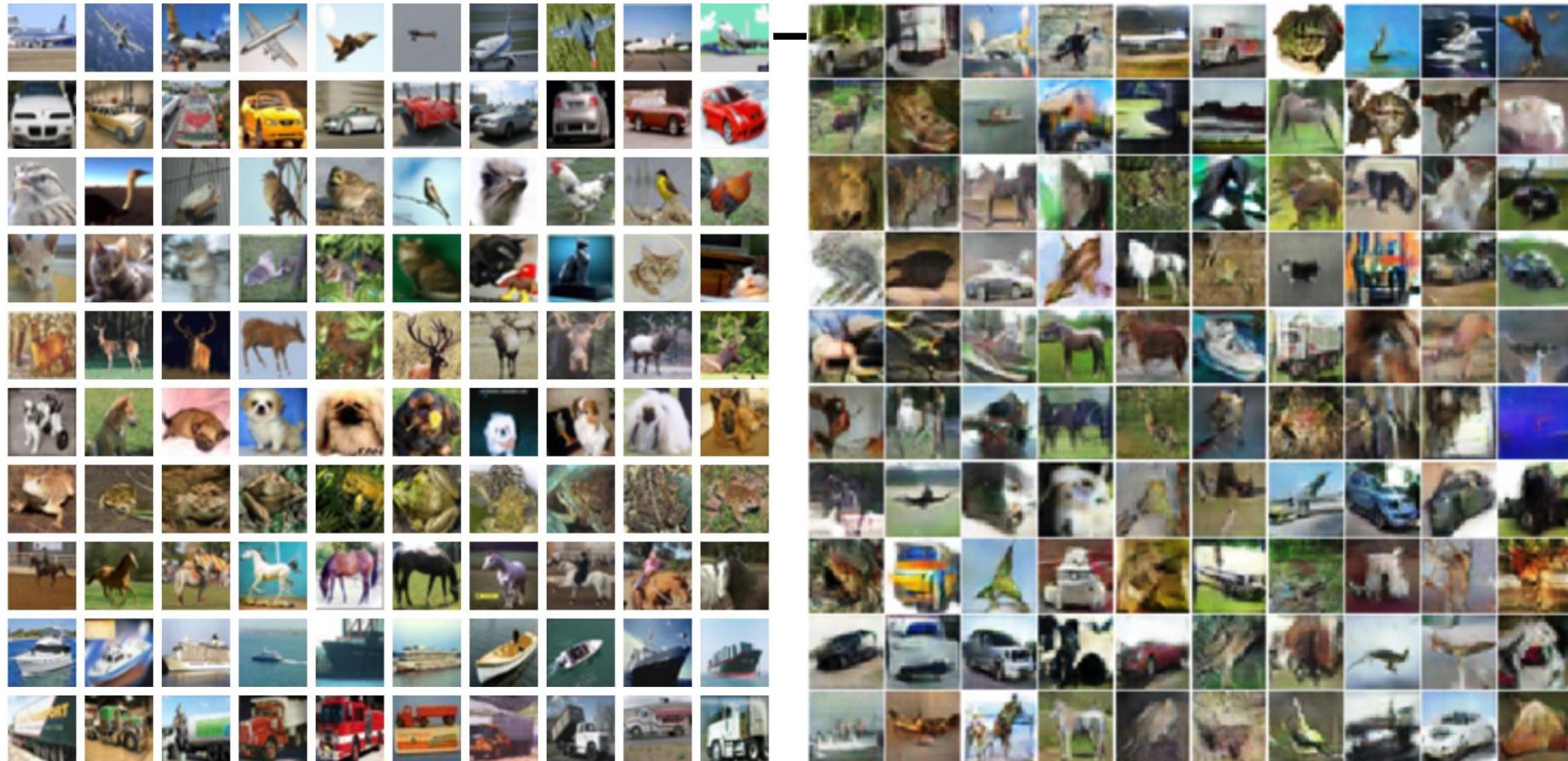
# *“Improved Techniques for Training GANs” by Salimans et. al*

---

One-sided Label smoothing - replaces the 0 and 1 targets for a classifier with smoothed values, like .9 or .1 to reduce the vulnerability of neural networks to adversarial examples.

Virtual batch Normalization - each example  $x$  is normalized based on the statistics collected on a reference batch of examples that are chosen once and fixed at the start of training, and on  $x$  itself.

---



Original CIFAR-10 vs. Generated CIFAR-10 samples

Source: "Improved Techniques for Training GANs" <https://arxiv.org/abs/1606.03498>



# Variations to GANs

---

Several new concepts built on top of GANs have been introduced –  
InfoGAN – Approximate the data distribution and learn interpretable, useful vector representations of data.  
Conditional GANs - Able to generate samples taking into account external information (class label, text, another image). Force G to generate a particular type of output.

---



# Major Difficulties

---

Networks are difficult to converge.

Ideal goal – Generator and discriminator to reach some desired equilibrium but this is rare.

GANs are yet to converge on large problems (E.g. Imagenet).

---



# Common Failure Cases

---

The discriminator becomes too strong too quickly and the generator ends up not learning anything.

The generator only learns very specific weaknesses of the discriminator.

The generator learns only a very small subset of the true data distribution.

---



- 
- Normalize the inputs**
  - A modified loss function**
  - Use a spherical Z**
  - BatchNorm**
  - Avoid Sparse Gradients: ReLU, MaxPool**
  - Use Soft and Noisy Labels**
  - DCGAN / Hybrid Models**
  - Track failures early (D loss goes to 0: failure mode)**
  - If you have labels, use them**
  - Add noise to inputs, decay over time**
-



# Conclusions

---

Train GAN – Use discriminator as base model for transfer learning and the fine-tuning of a production model.

A well-trained generator has learned the true data distribution well - Use generator as a source of data that is used to train a production model.

---



# Dive Deeper?

---

# Generative Adversarial Networks (GANs)

Ian Goodfellow, OpenAI Research Scientist  
NIPS 2016 tutorial  
Barcelona, 2016-12-4

Ian Goodfellow's NIPS 2016 Tutorial

Available online.

OpenAI

---



# References

---

<https://tryolabs.com/blog/2016/12/06/major-advancements-deep-learning-2016/>

<https://blog.waya.ai/introduction-to-gans-a-boxing-match-b-w-neural-nets-b4e5319cc935#.6l7zh8u50>

[https://en.wikipedia.org/wiki/Generative\\_adversarial\\_networks](https://en.wikipedia.org/wiki/Generative_adversarial_networks)

<http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/>

<https://github.com/soumith/ganhacks>



# Unsupervised Learning

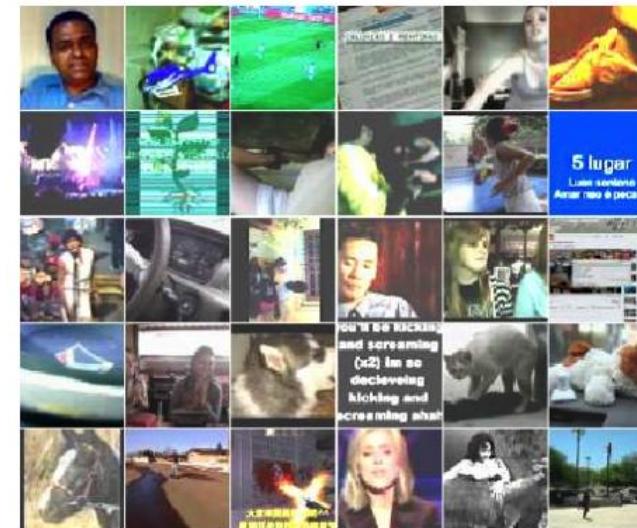
---

# Discovering High-Level Features Via Unsupervised Learning

(Le et al., 2012)

Training set  
10M YouTube videos  
single frame sampled  
from each

200 x 200 pixels  
fewer than 3% of  
frames contain  
faces (using OpenCV  
face detector)





## Sparse deep encoder architecture

3 encoding layers

each “layer” consists of three transforms

spatially localized receptive fields to detect features

spatial pooling to get translation invariance

subtractive/divisive normalization to get sparse activation patterns

not convolutional

1 billion weights

train with version of sparse PCA

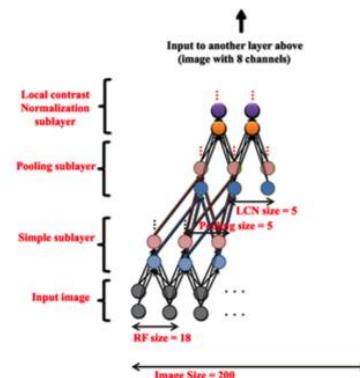
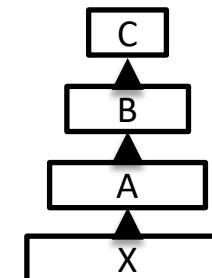


Figure 9. Diagram of the network we used with more detailed connectivity patterns. Color arrows mean that weights only connect to only one map. Dark arrows mean that weights connect to all maps. Pooling neurons only connect to one map whereas simple neurons and LCN neurons connect to all maps.



# Some Neurons Become Face Detectors

---

Look at all neurons in final layer and find the best face detector



*Figure 3.* Top: Top 48 stimuli of the best neuron from the test set. Bottom: The optimal stimulus according to numerical constraint optimization.

---



# Some Neurons Become Cat and Body Detectors

---



# How Fancy Does Unsupervised Learning Have To Be?

---

Coates, Karpathy, Ng (2012)

K-means clustering – to detect

prototypical features

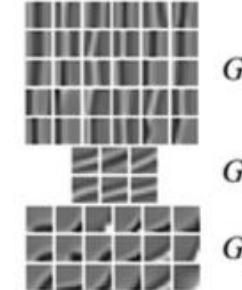
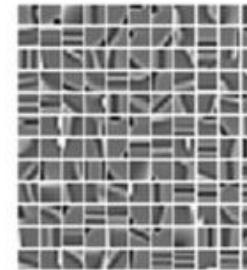
agglomerative clustering – to

pool together features that

are similar under transformation

multiple stages

Face detectors emerge



# Autoencoders

---

Self-supervised training procedure

Given a set of input vectors (no target outputs)

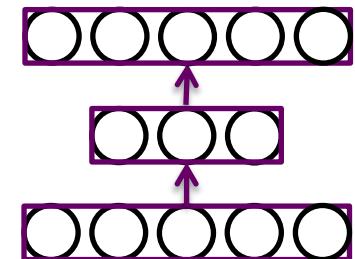
Map input back to itself via a hidden layer bottleneck

How to achieve bottleneck?

- Fewer neurons

- Sparsity constraint

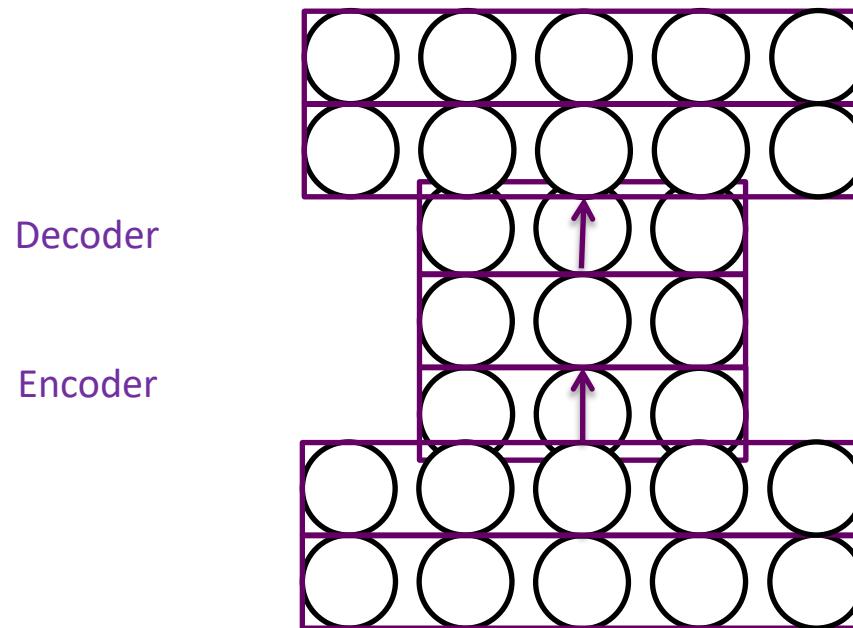
- Information transmission constraint (e.g., add noise to unit, or shut off randomly, a.k.a. dropout)



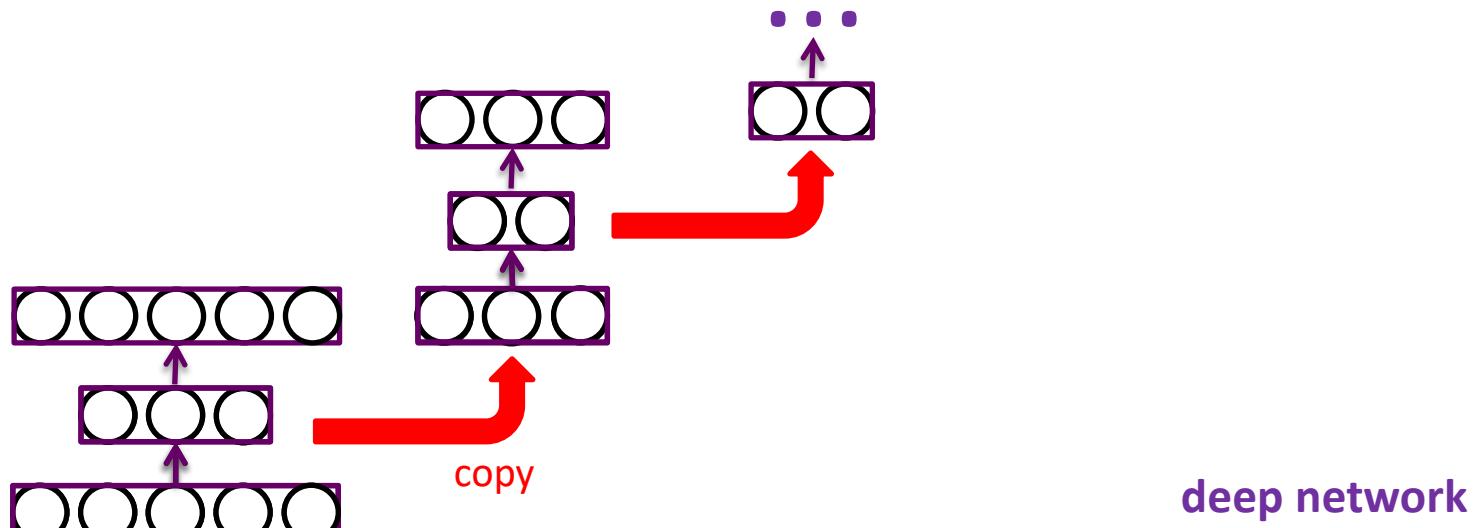


# Autoencoder Combines An Encoder And A Decoder

---



# Stacked Autoencoders



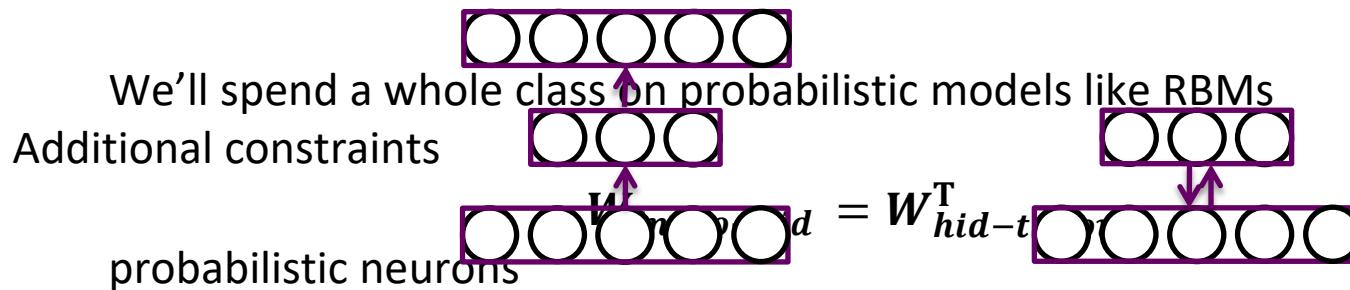
Note that decoders can be stacked to produce a generative domain model



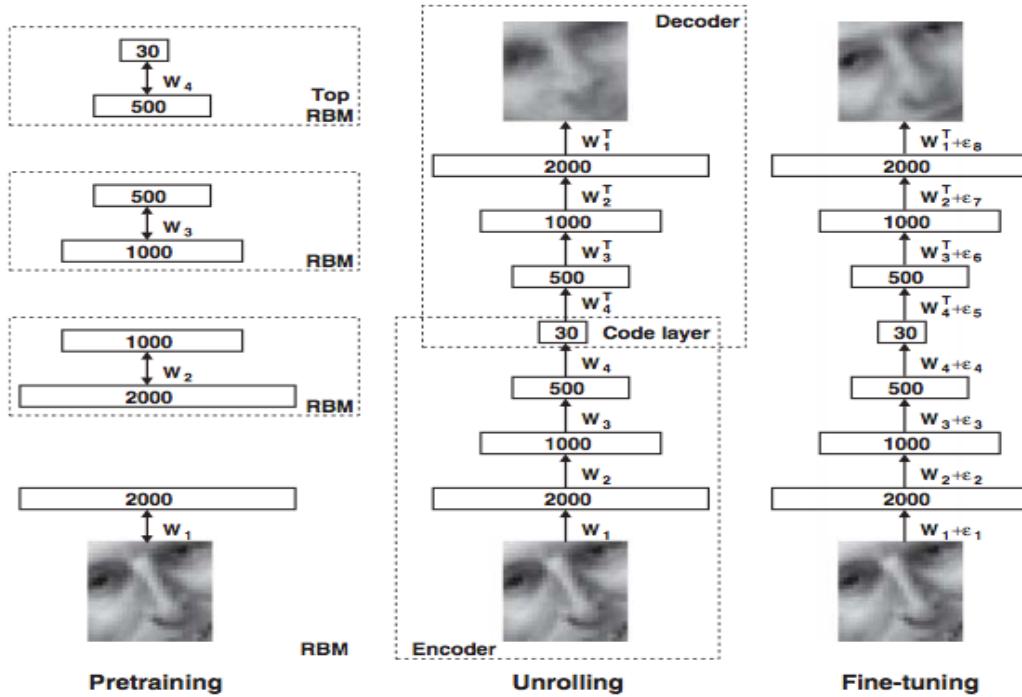
# Restricted Boltzmann Machines (RBMs)

---

For today, RBM is like an autoencoder with the output layer folded back onto the input.



# Deep RBM Autoencoder

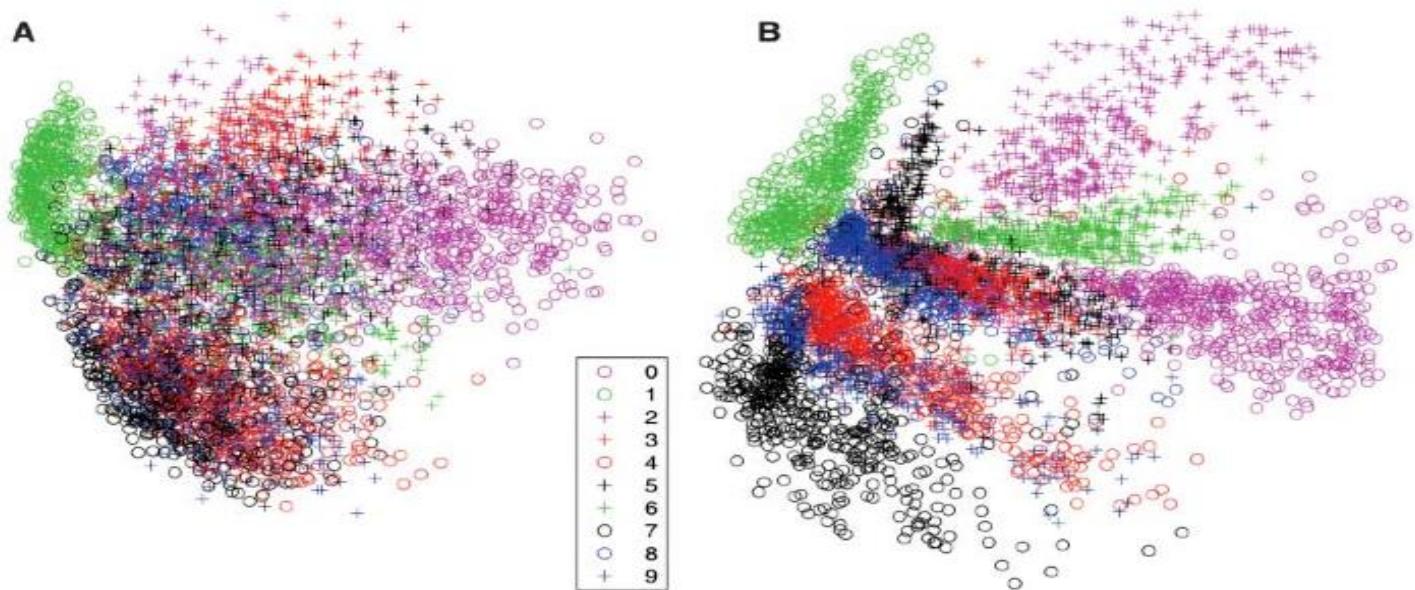


**Fig. 1.** Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the “data” for training the next RBM in the stack. After the pretraining, the RBMs are “unrolled” to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

Hinton & Salakhutdinov (2006)



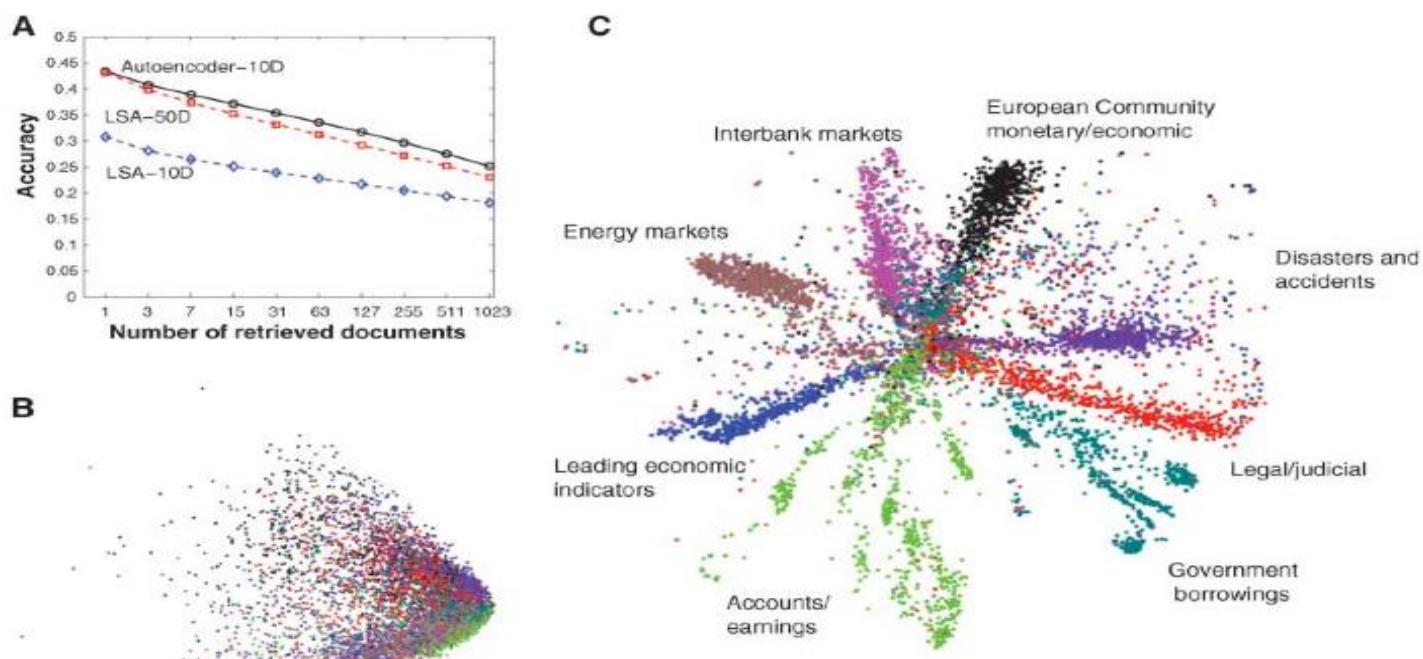
**Fig. 3.** (A) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (B) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (8).



○	0
○	1
+	2
+	3
○	4
+	5
○	6
○	7
○	8
+	9



**Fig. 4.** (A) The fraction of retrieved documents in the same class as the query document when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (B) The codes produced by two-dimensional LSA. (C) The codes produced by a 2000-500-250-125-2 autoencoder.



# Why Does Unsupervised Pretraining Help Deep Learning?

(Erhan, Bengio, Courville, Manzagol, Vincent & Bengio 2010)

More hidden layers ->  
increase likelihood of  
poor local minima  
result is robust to random  
initializatio

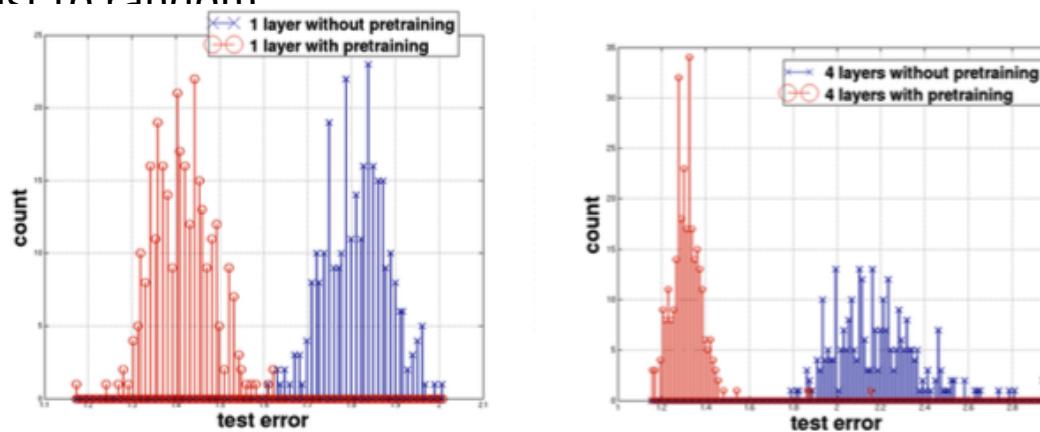


Figure 2: Histograms presenting the test errors obtained on MNIST using models trained with or without pre-training (400 different initializations each). **Left:** 1 hidden layer. **Right:** 4 hidden layers.

# Visualizing Learning Trajectory

Train 50 nets with and without pretraining on MNIST

No easy way to compare weights, but it is possible to compare  
*functions*

Comparison procedure

At various points in training, form giant vector of output for each  
training example

Perform dimensionality reduction using ISOMAP

Observations

Pretrained and not-pretrained models start and stay in different regions of function space

More variance (different local optima?) with not-pretrained models

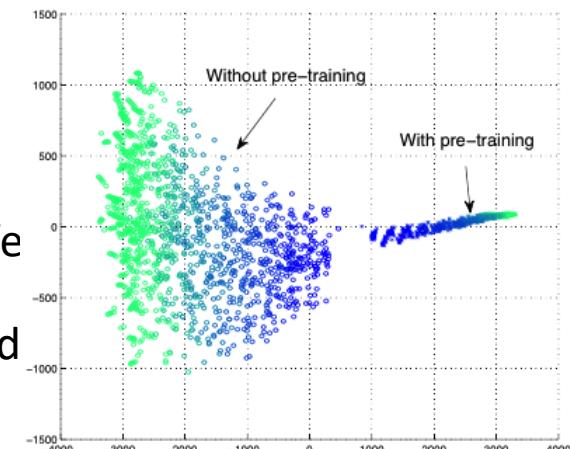


Figure 6: 2D visualization with ISOMAP of the functions represented by 50 networks with and 50 networks without pre-training, as supervised training proceeds over MNIST. See Section 6.3 for an explanation. Color from dark blue to cyan indicates a progression in training iterations (training is longer without pre-training). The plot shows models with 2 hidden layers but results are similar with other depths.



# Using Autoencoders To Initialize Weights For Supervised Learning

---

Effective pretraining of weights should act as a regularizer

Limits the region of weight space that will be explored by supervised learning  
Autoencoder must be learned well enough to move weights away from origin

# The Danger Of Unsupervised Pretraining: Simple Autoencoder

## Architecture

$$y = \tanh(w_2 h)$$

$$h = \tanh(w_1 x)$$

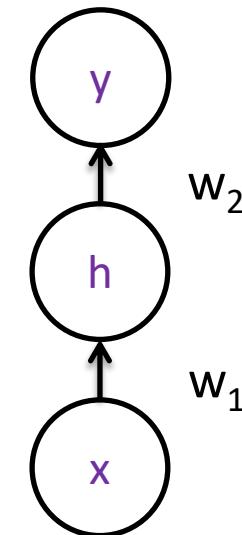
100 training examples with

$$x \sim Uniform(-1, +1)$$

$$y_{target} = x$$

What are the optimal weights?

How many solutions are there?





# Simple Autoencoder: Weight Search

---

```
function wtspace()

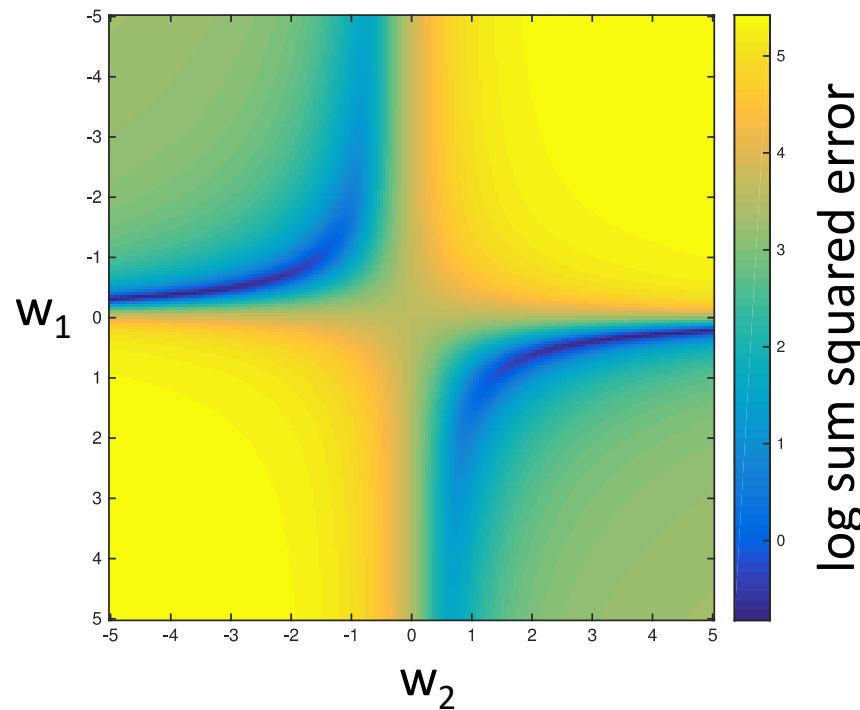
    train = rand(1,100)*2-1;
    wrange = -5:.05:5;
    for w1=1:length(wrange)
        for w2=1:length(wrange)
            savew(w1,w2) = compute_err(train, train, wrange(w1), wrange(w2));
        end
    end
    imagesc(wrange, wrange, savew)
    axis square
    colorbar
    set(gca, 'xtick', -5:5)
    set(gca, 'ytick', -5:5)
    set(gcf, 'paperpositionmode', 'auto')
    print2pdf autoencoder.pdf

function e = compute_err(in, out,w1,w2)
    h = tanh(in .* w1);
    o = tanh(h .* w2);
    e = sqrt(sum((out-o).^2));
```



# Simple Autoencoder: Error Surface

---



# Simple Supervised Problem

## Architecture

$$y = \tanh(w_2 h + 1)$$

$$h = \tanh(w_1 x - 1)$$

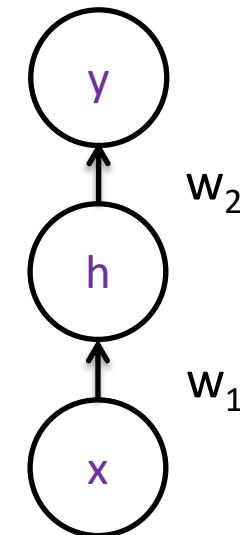
100 training examples with

$$x \sim \text{Uniform}(-1, +1)$$

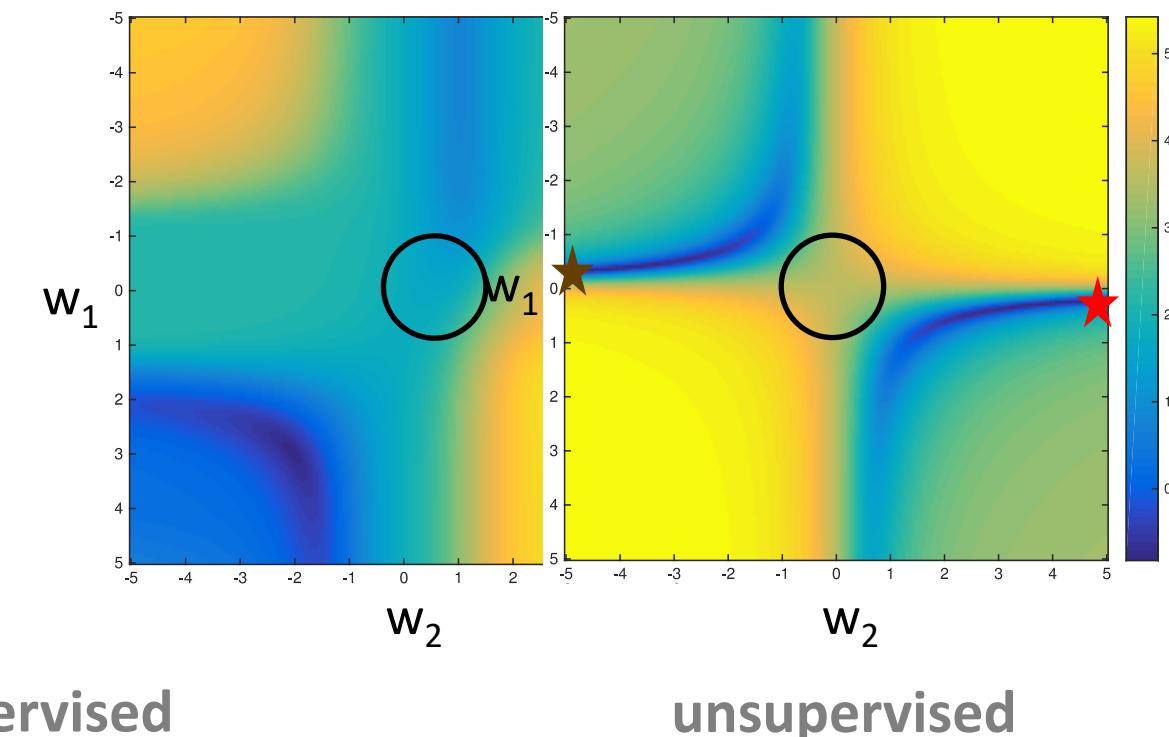
$$y_{target} = \tanh(-2 \tanh(3x + 1) - 1)$$

What are the optimal weights?

How many solutions are there?

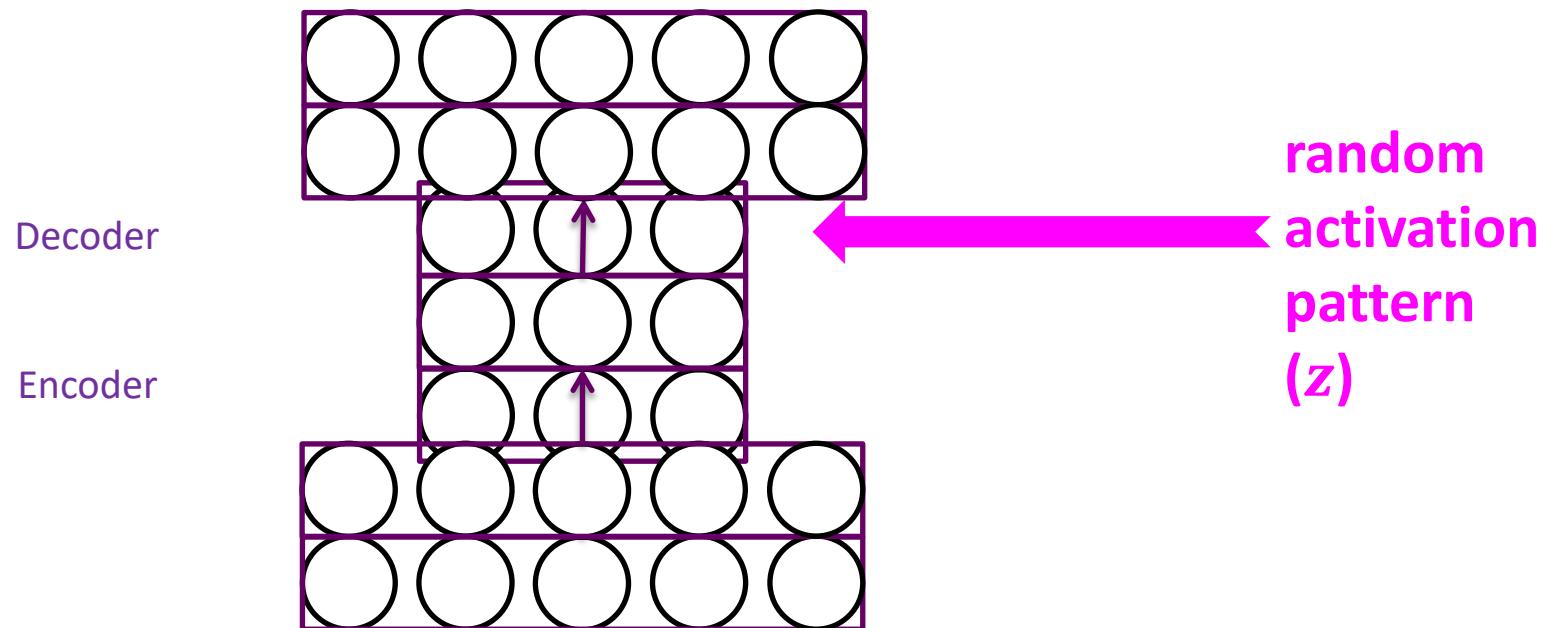


# Simple Supervised Problem: Error Surface



# Variational Autoencoders For Pattern Synthesis

Suppose we want to use an autoencoder to generate images.

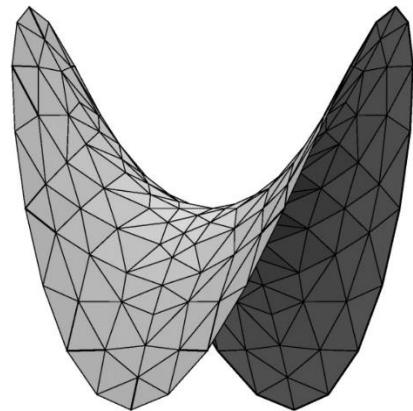


# Problem With Using Decoder For Generation

---

Because we don't know the distribution of data in the latent space...

No assurance that random activati



... will produce a sensible output

# Reinterpreting Decoder As Probabilistic Model

## Notation

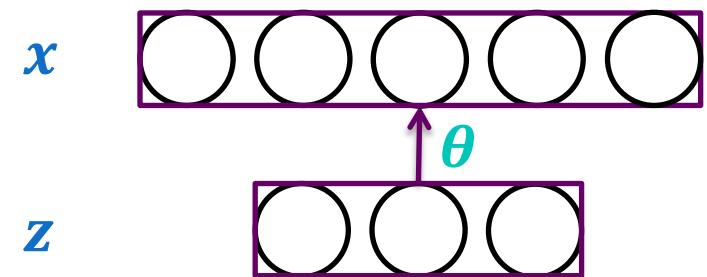
$x$ : input vector

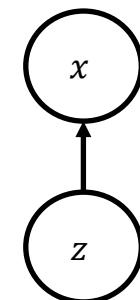
$z$ : hidden (latent) vector

$\theta$ : parameters of decoder

(1) draw a  $z$  vector from prior  $p(z)$

(2) draw  $x$  given  $z$  from  $p(x|z)$




$$p_{\theta}(x, z) = p(z)p_{\theta}(x|z)$$
$$p_{\theta}(x|z) = \mathcal{N}(x; f_{\theta}(z), \sigma^2 I)$$
$$p(z) = \mathcal{N}(z; \mathbf{0}, I)$$



# How Do We Train Decoder?

---

To use an algorithm like EM, we must infer  $\mathbf{z}$

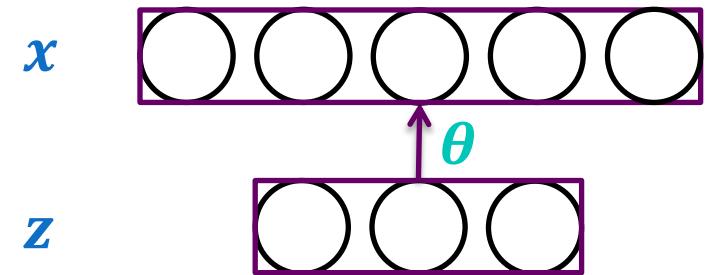
$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{z})}{\int_{\mathbf{z}'} p(\mathbf{x}, \mathbf{z}')}$$

Intractable in a neural net due to normalization constant

Alternative

Approximate posterior with another neural net which computes  $q_\phi(\mathbf{z}|\mathbf{x})$

This is just a probabilistic encoder with variational parameters  $\phi$





# Adding the Encoder

---

Encoder computes  $q_\phi(z|x)$

How do we make neural net response nondeterministic?

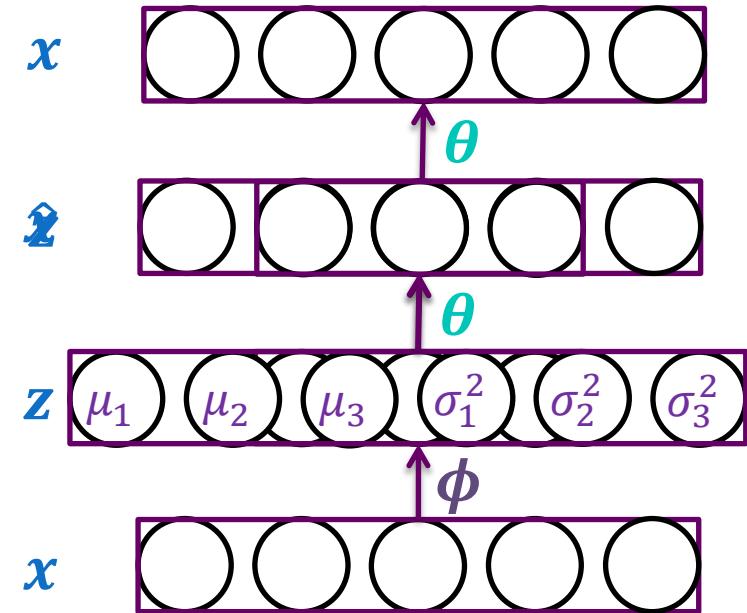
How do we propagate a probability distribution through neural net?

**Sampling:**  $\hat{z}_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$

How do we ensure that

$$q(z|x) \approx \frac{p(x,z)}{\int_z p(x,z')} ?$$

Maximize the ELBO





# Optimization problem

---

Objective function

$$L_{\phi,\theta} = E_{z \sim q_\phi} \log p_\theta(x|z) - D_{KL}(q_\phi(z|x) \| p(z))$$

This is a variational lower bound

$$L_{\phi,\theta} \leq \log p_\theta(x)$$

Specifically:  $L_{\phi,\theta} = \log p_\theta(x) - D_{KL}(q_\phi(z|x) \| p_\theta(z|x))$

maximizing  $L_{\phi,\theta}$  is a proxy to making  $q_\phi(z|x)$  match  $p_\theta(z|x)$

---



---

Our goal is to have a generative model that has a high likelihood of producing the observed data

$$p(x) = \int p_{\theta}(x|z)p(z)dz \cong \int p_{\theta}(x|z)q_{\phi}(z|x)dz = E_{z \sim q_{\phi}} p(x|z)$$

Achieved by maximizing

$$\begin{aligned} L_{\phi, \theta} &= E_{z \sim q_{\phi}} p_{\theta}(x|z) - D_{KL}(q_{\phi}(z|x) \| p(z)) \\ &= E_{z \sim q_{\phi}} p_{\theta}(x|z) - E_{z \sim q_{\phi}} [\log q_{\phi}(z|x) - \log p(z)] \end{aligned}$$

Last detail

Approximate the expectation with samples



# Complete Model

---

$$p_{\theta}(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$$

$$p_{\theta}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}), \sigma(\mathbf{z})\mathbf{I})$$

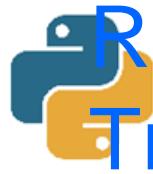
$$q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \sigma(\mathbf{x})\mathbf{I})$$

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}, \phi; \mathbf{x}) = \underbrace{\log p_{\theta}(\mathbf{x}|\mathbf{z}^{(l)})}_{\text{(noisy) negative reconstruction error}} + \underbrace{\log p_{\theta}(\mathbf{z}^{(l)}) - \log q_{\phi}(\mathbf{z}^{(l)}|\mathbf{x})}_{\text{regularization terms}}$$

where  $\mathbf{z}^{(l)} \sim q_{\phi}(\mathbf{z}|\mathbf{x})$

From Kingma ICLR 2014 Slides

---



# Random Samples From Generative Model Trained on *Faces in the Wild*

---



From Kingma  
ICLR 2014  
slides

---



## [VAE faces](#)

## [VAE faces demo](#)

## [VAE MNIST](#)

## [VAE street addresses](#)



# Contractive Autoencoders

---

A good hidden representation will be insensitive to most small changes in the input (while still preserving information to reconstruct the input)

Objective function

Frobenius norm of the Jacobian

For logistic hidden:

$$\|J_f(x)\|_F^2 = \sum_{ij} \left( \frac{\partial h_j(x)}{\partial x_i} \right)^2 \quad \mathcal{J}_{\text{CAE}}(\theta) = \sum_{x \in D_n} (L(x, g(f(x))) + \lambda \|J_f(x)\|_F^2)$$

$$\|J_f(x)\|_F^2 = \sum_{i=1}^{d_h} (h_i (1 - h_i))^2 \sum_{j=1}^{d_x} W_{ij}^2$$

Tries to make the hidden units saturate, especially where input-to-hidden weights are large



# Sparse Encodings

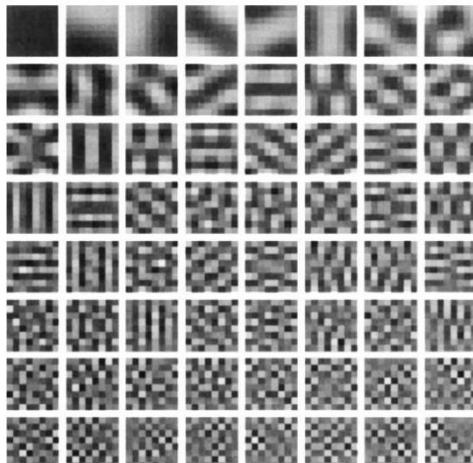
---

We may wish to have few units in the hidden bottleneck representation for any input

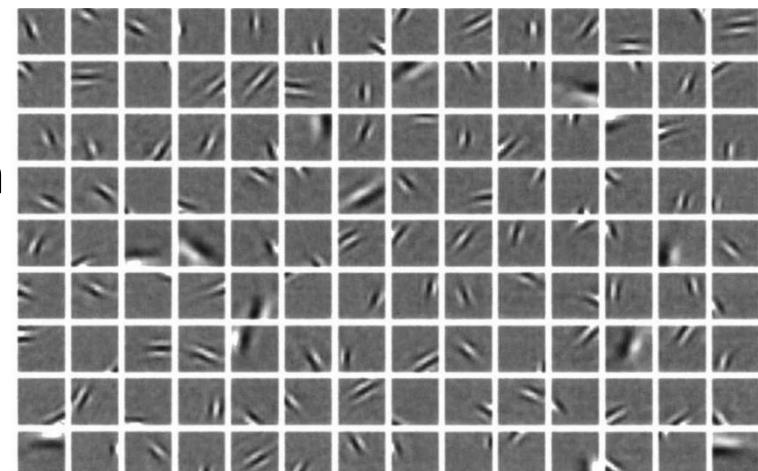
Biological motivation

E.g., unsupervised training on natural image patches

PCA



Olshausen  
& Field  
(1996)





# Sparse Encoding

---

Loss function has two components

information preservation

- input reconstruction in autoencoder

- mutual information in simple encoder

sparsity constraint

$$\mathcal{L} = \mathcal{L}_{\text{preservation}} + \lambda \mathcal{L}_{\text{sparsity}}$$

---

# Examples of Sparsity Constraints

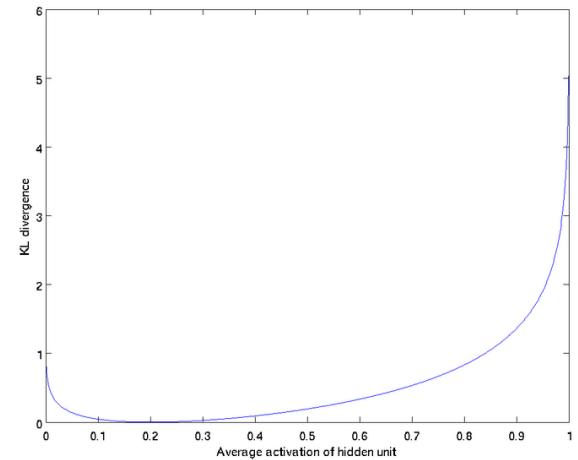
Olshausen & Field (1996)

$$\begin{aligned}\mathcal{L}_{\text{sparsity}} &= -e^{-h^2} \\ \mathcal{L}_{\text{sparsity}} &= \ln(1 + h^2) \\ \mathcal{L}_{\text{sparsity}} &= |h|\end{aligned}$$

Many other possibilities

If  $h \in [0, 1]$  and  $h^*$  is target sparsity,

$$\mathcal{L}_{\text{sparsity}} = \text{KL}(h^* || h) = h^* \log\left(\frac{h^*}{h}\right) + (1 - h^*) \log\left(\frac{1 - h^*}{1 - h}\right)$$

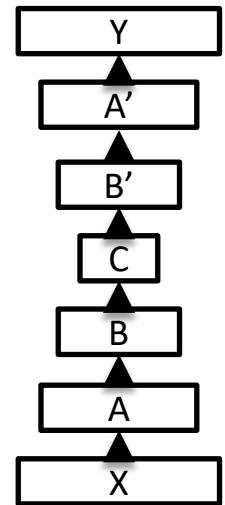


# Discovering Binary Codes

If hidden units of autoencoder discover binary code, we can measure information content in hidden representations interface with digital representations

One scheme that does not work terribly well is the addition of a cost term on the hidden units that penalizes nonbinary hidden activations

$$\mathcal{L} = \mathcal{L}_{\text{reconstruction}} + \lambda \sum_i h_i(1 - h_i)$$





# Discovering Binary Codes

---

Make network robust to noise in hidden units

Scheme 1

add noise to net input to bottleneck layer

Scheme 2

treat  $h \in [0, 1]$  as a probability of turning the unit on or off during activation propagation

during back propagation, use continuous value of  $h$

Videos

[Hinton video 1](#)

[Hinton video 2](#)

---

# Denoising Autoencoders

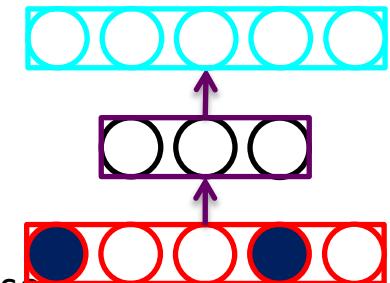
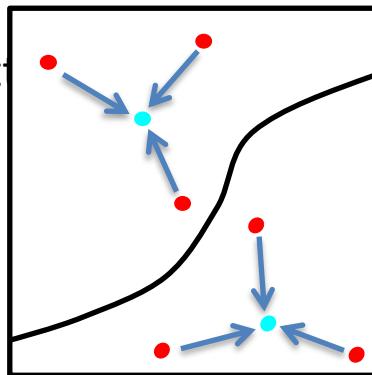
Randomly set inputs to zero (like dropout on inputs)

Target output has missing features filled in

Visualization

related to  
attractor nets

Removing features between layers with additive Gaussian noise



# Attractor Networks For Pattern Completion

The Jets and The Sharks

Name	Gang	Age	Edu	Mar	Occupation
Art	Jets	40's	J.H.	Sing.	Pusher
Al	Jets	30's	J.H.	Mar.	Burglar
Sam	Jets	20's	COL.	Sing.	Bookie
Clyde	Jets	40's	J.H.	Sing.	Bookie
Mike	Jets	30's	J.H.	Sing.	Bookie
Jim	Jets	20's	J.H.	Div.	Burglar
Greg	Jets	20's	H.S.	Mar.	Pusher
John	Jets	20's	J.H.	Mar.	Burglar
Doug	Jets	30's	H.S.	Sing.	Bookie
Lance	Jets	20's	J.H.	Mar.	Burglar
George	Jets	20's	J.H.	Div.	Burglar
Pete	Jets	20's	H.S.	Sing.	Bookie
Fred	Jets	20's	H.S.	Sing.	Pusher
Gene	Jets	20's	COL.	Sing.	Pusher
Ralph	Jets	30's	J.H.	Sing.	Pusher
Phil	Sharks	30's	COL.	Mar.	Pusher
Ike	Sharks	30's	J.H.	Sing.	Bookie
Nick	Sharks	30's	H.S.	Sing.	Pusher
Don	Sharks	30's	COL.	Mar.	Burglar
Ned	Sharks	30's	COL.	Mar.	Bookie
Karl	Sharks	40's	H.S.	Mar.	Bookie
Ken	Sharks	20's	H.S.	Sing.	Burglar
Earl	Sharks	40's	H.S.	Mar.	Burglar
Rick	Sharks	30's	H.S.	Div.	Burglar
Ol	Sharks	30's	COL.	Mar.	Pusher
Neal	Sharks	30's	H.S.	Sing.	Bookie
Dave	Sharks	30's	H.S.	Div.	Pusher

FIGURE 10. Characteristics of a number of individuals belonging to two gangs, the Jets and the Sharks. (From "Retrieving General and Specific Knowledge From Stored Knowledge of Specifics" by J. L. McClelland, 1981, *Proceedings of the Third Annual Conference of the Cognitive Science Society*, Berkeley, CA. Copyright 1981 by J. L. McClelland. Reprinted by permission.)

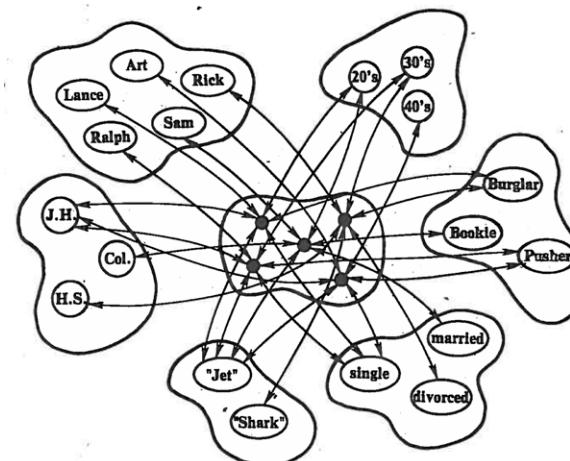


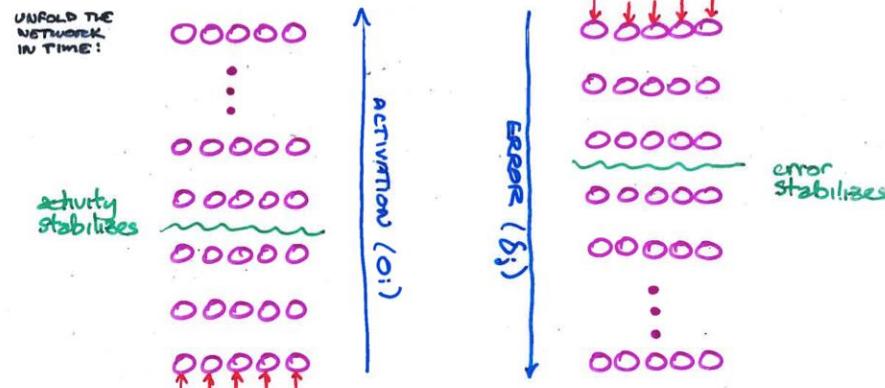
FIGURE 11. Some of the units and interconnections needed to represent the individuals shown in Figure 10. The units connected with double-headed arrows are mutually excitatory. All the units within the same cloud are mutually inhibitory. (From "Retrieving General and Specific Knowledge From Stored Knowledge of Specifics" by J. L. McClelland, 1981, *Proceedings of the Third Annual Conference of the Cognitive Science*



# Training Attractor Networks

Almeida /  
Pineda  
algorithm

- Recurrent network
- Probe via external inputs
- Allows network to settle
- No time-varying inputs



$$\begin{aligned}\Delta w_{ji} &= \varepsilon \sum_{n=1}^{\infty} o_i(n) \delta_j(n) \\ &= \varepsilon \left[ \sum_1 \hat{o}_i(n) \hat{\delta}_j + \sum_2 \hat{o}_i \hat{\delta}_j + \sum_3 \hat{o}_i(n) \delta_j(n) \right]\end{aligned}$$

But number of time steps in ① and ③ is finite; number of time steps in ② is infinite  $\Rightarrow$  ignore ① and ③

$$\Delta w_{ji} = \varepsilon \hat{o}_i \hat{\delta}_j \quad \text{where } \hat{o}_i \text{ is fixed point of activity; } \hat{\delta}_j \text{ is fixed point of error}$$



# Comments on Almeida / Pineda Algorithm

---

This method works only if network reaches a fixed (stable) point in both forward and backward phases

can show that if network is stable in forward direction, it is also stable in the backward direction.

Can show that network will be stable in the forward direction if weights are symmetric ( $w_{ij} = w_{ji}$ )

We'll see this with Hopfield network

Algorithm makes no guarantee about network dynamics

i.e., can't train the net to settle in a short amount of time

---



# Aditya's Image Superresolution

---

**Input**



**Enhanced**



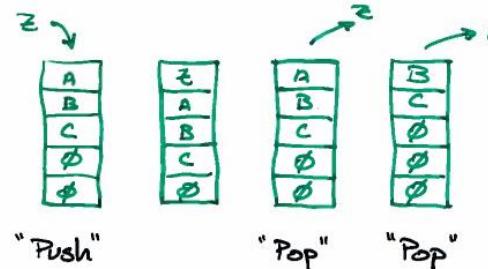
**Original**





# Recursive Neural Nets

POLACK (1991)  
Recursive Auto-Associative  
Memory (RAAM)  
Representing stacks



$\phi = \text{null}$   
(empty stack)

Use of stacks: Subroutines, Subgoals

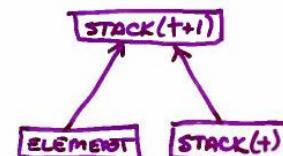
Another representation of stack: Right branching binary tree



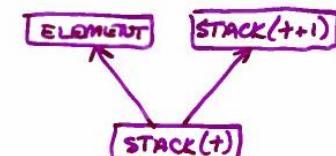
Implementing "push" and "pop" with connectionist hardware:

- set of units that represent stack state at time  $t$  +  $\text{stack}(t)$
- set of units that represent element to be pushed or popped  $\text{ELEMENT}$

Push Operation



Pop Operation





# Recursive Neural Nets

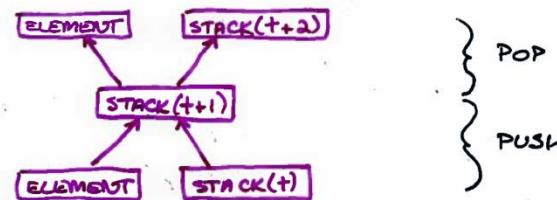
POLACK (1991)

Recursive Auto-Associative  
Memory (RAAM)

Representing stacks

How do we train connections to implement "push" and "pop"?

Combine the two operations in an autoencoder network:



Training examples

<u>operation</u>	<u>stack state</u>	<u>input / target output</u>	<u>hidden</u>
	$\emptyset$		
PUSH A	(A)	A + $\emptyset$	R <sub>A</sub>
PUSH B	(B A)	B + R <sub>A</sub>	R <sub>BA</sub>
PUSH C	(C B A)	C + R <sub>BA</sub>	R <sub>CBA</sub>
POP	(B A)		
PUSH D	(D B A)	D + R <sub>BA</sub>	R <sub>DBA</sub>
PUSH C	(C D B A)	C + R <sub>DBA</sub>	R <sub>CDBA</sub>
POP	(D B A)		
POP	(B A)		
POP	(A)		
PUSH D	(D A)	D + R <sub>A</sub>	R <sub>DA</sub>
POP	(A)		
POP	$\emptyset$		

Use hidden representation at one time as input (stack state) at the next time

Note: change in weights  $\rightarrow$  change in hidden unit rep  $\rightarrow$  change in input with small  $\epsilon$ , weights will converge (not guaranteed)  $\rightarrow$  hidden unit rep will converge



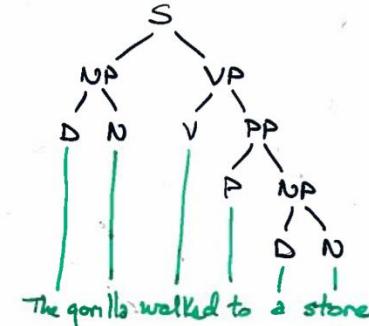
# Recursive Neural Nets

POLACK (1991)

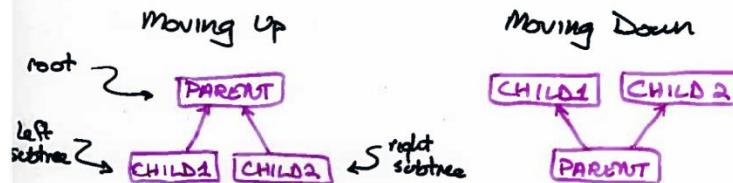
Recursive Auto-Associative

Memory (RAAM)

Representing binary trees

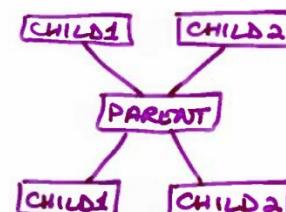


Need to be able to move up and down tree.



Same problem as stack except that, since both CHILD1 and CHILD2 can be PARENT, you need some number of units in each pool.

2K-K-2K autoencoder:





# Recursive Neural Nets

POLACK (1991)

Recursive Auto-Associative  
Memory (RAAM)  
Representing binary trees

Training sample for binary tree learning:

$((D \cdot N) \cdot (V \cdot (P \cdot (D \cdot N))))$

Tree	Input/Target output	Hidden
$(D \cdot N)$	$D + N$	$R_{DN}$
$(P \cdot (D \cdot N))$	$P + R_{DN}$	$R_{PDN}$
$(V \cdot (P \cdot (D \cdot N)))$	$V + R_{PDN}$	$R_{VPDN}$
$((D \cdot N) \cdot (V \cdot (P \cdot (D \cdot N))))$	$R_{DN} + R_{VPDN}$	$R_S$

Grammar:

$S \rightarrow NP VP \mid NP V$   
 $NP \rightarrow D AP \mid DN \mid NPP$   
 $PP \rightarrow P NP$   
 $VP \rightarrow V NP \mid V PP$   
 $AP \rightarrow A AP \mid AN$

Each terminal (D A N V & P) was represented as a 1-bit-in-5 code padded with 5 zeros. A 20-10-20 RAAM devised the representations shown in figure 3.

NP	(D N) 00000 · · · · ·
	(D (A (A N))) 00000 · · · · ·
	(D (A N)) 00000 · · · · ·
	(D (A) N) 00000 · · · · ·
VP	(V (P (D N))) 00000 · · · · ·
	(V (D (A N))) 00000 · · · · ·
	(V (A N)) 00000 · · · · ·
PP	(P (D N)) 00000 · · · · ·
	(P (D (A N))) 00000 · · · · ·
AP	(A N) 00000 · · · · ·
	(A (A N)) 00000 · · · · ·
	(A (A) N) 00000 · · · · ·
S	(D N) V 00000 · · · · ·
	((D N) (V (D (A N)))) 00000 · · · · ·
	((D (A N)) (V (P (D N)))) 00000 · · · · ·

Figure 3.

Representations of all the binary trees in the training set, devised by a 20-10-20 RAAM, manually clustered

**CLAIM** This network shows compositionality. Can take novel combinations of two old structures to form a new structure

Is this true? Does training generalize in this way? Not yet known

E.g., stack1 (A B C)  
 stack2 (D E F)  
 ↓  
 stack3 (A E F)



# Natural Language Processing

---



# Aspects of language processing

---

Word, lexicon: lexical analysis

Morphology, word segmentation

Syntax

Sentence structure, phrase, grammar, ...

Semantics

Meaning

Execute commands

Discourse analysis

Meaning of a text

Relationship between sentences (e.g. anaphora)

---



## Detect new words

Language learning

Machine translation

NL interface

Information retrieval

...



## 1950s

Early MT: word translation + re-ordering  
Chomsky's Generative grammar  
Bar-Hill's argument

## 1960-80s

### Applications

BASEBALL: use NL interface to search in a database on baseball games  
LUNAR: NL interface to search in Lunar  
ELIZA: simulation of conversation with a psychoanalyst  
SHREDLU: use NL to manipulate block world  
Message understanding: understand a newspaper article on terrorism  
Machine translation

### Methods

ATN (augmented transition networks): extended context-free grammar  
Case grammar (agent, object, etc.)  
DCG – Definite Clause Grammar  
Dependency grammar: an element depends on another

## 1990s-now

Statistical methods  
Speech recognition  
MT systems  
Question-answering  
...



# Classical symbolic methods

---

Morphological analyzer

Parser (syntactic analysis)

Semantic analysis (transform into a logical form, semantic network, etc.)

Discourse analysis

Pragmatic analysis



# Morphological analysis

---

Goal: recognize the word and category

Using a dictionary: word + category

Input form (*computed*)

Morphological rules:

Lemma + ed -> Lemma + e                    (verb in past form)

...

Is Lemma in dict.? If yes, the transformation is possible

Form -> a set of possible lemmas

---



# Parsing (in DCG)

s --> np, vp.

np --> det, noun.

np --> proper\_noun.

vp --> v, ng.

vp --> v.

[mary].

det -->[a]. det --> [an].

det --> [the].

noun --> [apple].

noun --> [orange].

proper\_noun --> [john].

proper\_noun -->

v --> [eats].

v --> [loves].

apple.

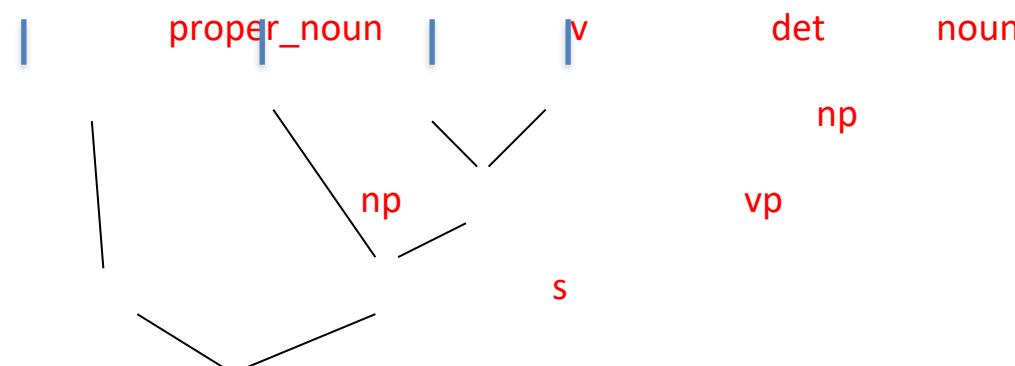
Eg.

john

eats

an

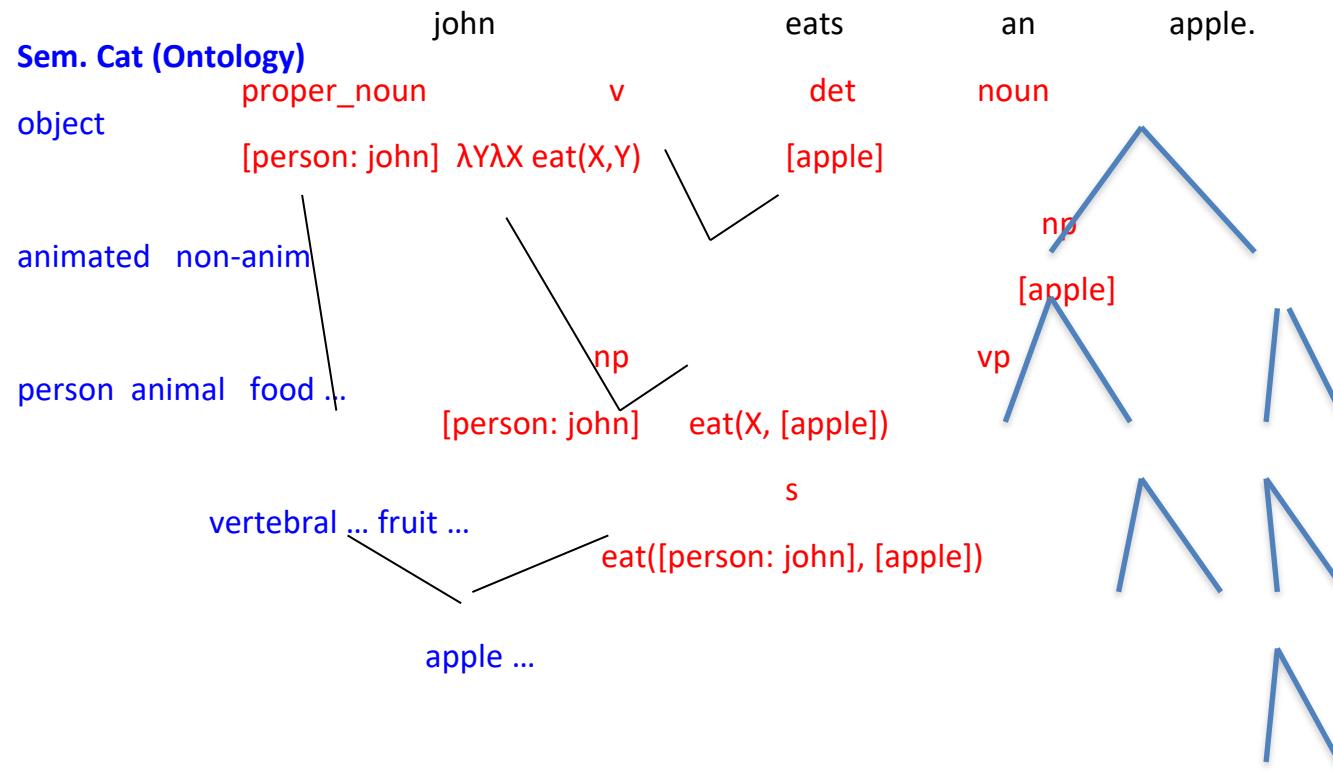
apple.





# Semantic analysis

---





# Parsing & semantic analysis

---

Rules: syntactic rules or semantic rules

What component can be combined with what component?

What is the result of the combination?

Categories

Syntactic categories: Verb, Noun, ...

Semantic categories: Person, Fruit, Apple, ...

Analyses

Recognize the category of an element

See how different elements can be combined into a sentence

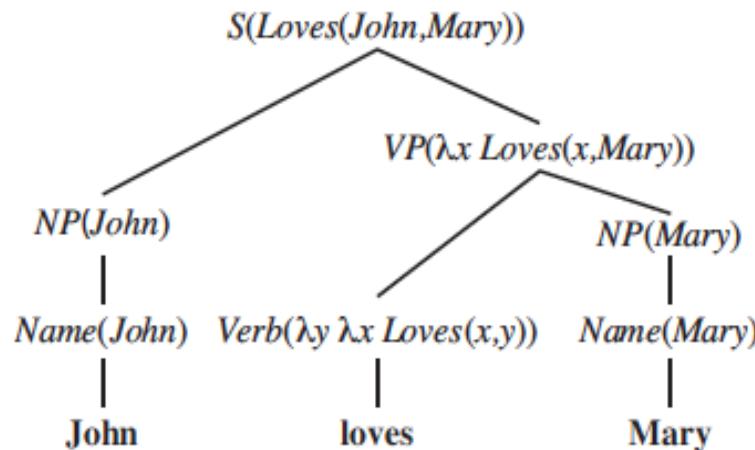
Problem: The choice is often not unique

---



# Write a semantic analysis grammar

$S(\text{pred}(\text{obj})) \rightarrow \text{NP}(\text{obj}) \text{ VP}(\text{pred})$   
 $\text{VP}(\text{pred}(\text{obj})) \rightarrow \text{Verb}(\text{pred}) \text{ NP}(\text{obj})$   
 $\text{NP}(\text{obj}) \rightarrow \text{Name}(\text{obj})$   
 $\text{Name}(\text{John}) \rightarrow \text{John}$   
 $\text{Name}(\text{Mary}) \rightarrow \text{Mary}$   
 $\text{Verb}(\lambda y \lambda x \text{ Loves}(x,y)) \rightarrow \text{loves}$





## Anaphora

He hits the car with a stone. **It** bounces back.

## Understanding a text

Who/when/where/what ... are involved in an event?

How to connect the semantic representations of different sentences?

What is the cause of an event and what is the consequence of an action?

...



## Practical usage of language: what a sentence means in practice

Do you have time?

How do you do?

It is too cold to go outside!

...



# Some success stories - ELIZA

---

**young woman:** Men are all alike.

**eliza:** In what way?

**young woman:** They're always bugging us about something specific or other.

**eliza:** Can you think of a specific example?

**young woman:** Well, my boyfriend made me come here.

**eliza:** Your boyfriend made you come here?

**young woman:** He says I'm depressed much of the time.

**eliza:** I am sorry to hear you are depressed.

**young woman:** It's true. I'm unhappy.

**eliza:** Do you think coming here will help you not to be unhappy?

**young woman:** I need some help, that much seems certain.

**eliza:** What would it mean to you if you got some help?

**young woman:** Perhaps I could learn to get along with my mother.

**eliza:** Tell me more about your family.

**young woman:** My mother takes care of me.

---



# ELIZA – Cont'd

---

**eliza:** Who else in your family takes care of you?

**young woman:** My father.

**eliza:** Your father.

**young woman:** You are like my father in some ways.

**eliza:** What resemblance do you see?

**young woman:** You are not very aggressive but I think you don't want me to notice that.

**eliza:** What makes you think I am not very aggressive?

**young woman:** You don't argue with me.

**eliza:** Why do you think I don't argue with you?

**young woman:** You are afraid of me.

**eliza:** Does it please you to believe I am afraid of you?

**young woman:** My father is afraid of everybody.

**eliza:** What else comes to mind when you think of your father?

**young woman:** Bullies.

**eliza:** Does that have anything to do with the fact that your boyfriend made you come here?

---



# Methods used in Eliza

---

Find a trigger in the input of the user (e.g. father)

Evoke a possible candidate pattern (e.g. family or mother)  
(~limited parsing)

Compose a sentence by filling in the slots of the pattern (picking  
some elements from the user input)

If no appropriate pattern is found, ask a general question, possibly  
related to the user input

---



## RACTER – poem and prose composer

---

Slowly I dream of flying. I observe turnpikes and streets  
studded with bushes. Coldly my soaring widens my awareness.  
To guide myself I determinedly start to kill my pleasure  
during the time that hours and milliseconds pass away. Aid me in this  
and soaring is formidable, do not and singing is unhinged.

\*\*\*

Side and tumble and fall among  
The dead. Here and there  
Will be found a utensil.



# Success story – METEO

---

## Environment Canada

- Generate and translate METEO forecasts automatically English<->French

Aujourd'hui, 26 novembre

Généralement nuageux. Vents du sud-ouest de 20 km/h avec rafales à 40 devenant légers cet après-midi.

Températures stables près de plus 2.

Ce soir et cette nuit, 26 novembre

Nuageux. Neige débutant ce soir.

Accumulation de 15 cm. Minimum zéro.

Today, 26 November

Mainly cloudy. Wind southwest 20 km/h gusting to 40 becoming light this afternoon. Temperature steady near plus 2.

Tonight, 26 November

Cloudy. Snow beginning this evening. Amount 15 cm. Low zero.



---

## Ambiguity

Lexical/morphological: change (V,N), training (V,N), even (ADJ, ADV) ...

Syntactic: Helicopter powered by human flies

Semantic: He saw a man on the hill with a telescope.

Discourse: anaphora, ...

## Classical solution

Using a later analysis to solve ambiguity of an earlier step

Eg. He gives him the change.

(change as verb does not work for parsing)

He changes the place.

(change as noun does not work for parsing)

However: He saw a man on the hill with a telescope.

Correct multiple parsings

Correct semantic interpretations -> semantic ambiguity

Use contextual information to disambiguate (does a sentence in the text mention that "He" holds a telescope?)

---



# Rules vs. statistics

---

Rules and categories do not fit a sentence equally

Some are more likely in a language than others

E.g.

hardcopy: noun or verb?

$$P(N \mid \text{hardcopy}) \gg P(V \mid \text{hardcopy})$$

the training ...

$$P(N \mid \text{training, Det}) > P(V \mid \text{training, Det})$$

Idea: use statistics to help

---



# Statistical analysis to help solve ambiguity

Choose the most likely solution

$$\text{solution}^* = \operatorname{argmax}_{\text{solution}} P(\text{solution} \mid \text{word, context})$$

e.g.  $\operatorname{argmax}_{\text{cat}} P(\text{cat} \mid \text{word, context})$   
 $\operatorname{argmax}_{\text{sem}} P(\text{sem} \mid \text{word, context})$

Context varies largely (precedent word, following word, category of the precedent word, ...)

How to obtain  $P(\text{solution} \mid \text{word, context})$ ?

Training corpus

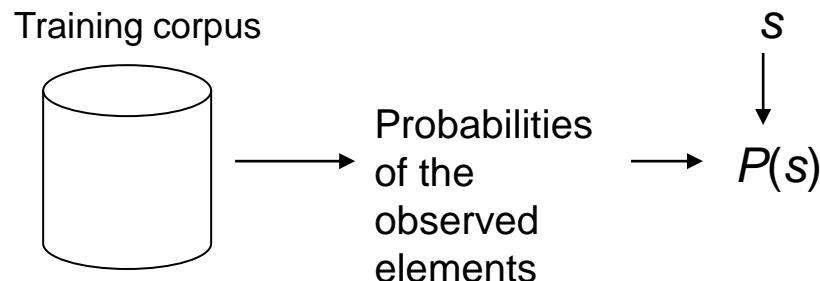
---



# Statistical language modeling

---

Goal: create a statistical model so that one can calculate the probability of a sequence of tokens  $s = w_1, w_2, \dots, w_n$  in a language.  
General approach:





# Prob. of a sequence of words

---

$$\begin{aligned} P(s) &= P(w_1, w_2, \dots, w_n) \\ &= P(w_1)P(w_2 | w_1) \dots P(w_n | w_{1,n-1}) \\ &= \prod_{i=1}^n P(w_i | h_i) \end{aligned}$$

Elements to be estimated:  $P(w_i | h_i) = \frac{P(h_i w_i)}{P(h_i)}$

- If  $h_i$  is too long, one cannot observe  $(h_i, w_i)$  in the training corpus, and  $(h_i, w_i)$  is hard to generalize
  - Solution: limit the length of  $h_i$
-



---

Limit  $h_i$  to n-1 preceding words  
Most used cases

Uni-gram:

$$P(s) = \prod_{i=1}^n P(w_i)$$

Bi-gram:

$$P(s) = \prod_{i=1}^n P(w_i | w_{i-1})$$

Tri-gram:

$$P(s) = \prod_{i=1}^n P(w_i | w_{i-2}w_{i-1})$$



## A simple example

(corpus = 10 000 words, 10 000 bi-grams)

$w_i$	$P(w_i)$	$w_{i-1}$	$w_{i-1}w_i$	$P(w_i w_{i-1})$
I (10)	10/10 000 = 0.001	# (1000)	(# I) (8)	8/1000 = 0.008
		that (10)	(that I) (2)	0.2
talk (8)	0.0008	I (10)	(I talk) (2)	0.2
		we (10)	(we talk) (1)	0.1
		...		
talks (8)	0.0008	he (5)	(he talks) (2)	0.4
		she (5)	(she talks) (2)	0.4
		...		
she (5)	0.0005	says (4)	(she says) (2)	0.5
		laughs (2)	(she laughs) (1)	0.5
		listens (2)	(she listens) (2)	1.0

Uni-gram:  $P(I, \text{talk}) = P(I) * P(\text{talk}) = 0.001 * 0.0008$

$P(I, \text{talks}) = P(I) * P(\text{talks}) = 0.001 * 0.0008$

Bi-gram:  $P(I, \text{talk}) = P(I | \#) * P(\text{talk} | I) = 0.008 * 0.2$

$P(I, \text{talks}) = P(I | \#) * P(\text{talks} | I) = 0.008 * 0$



---

History: short long

---

**modeling:** coarse refined

**Estimation:** easy difficult

Maximum likelihood estimation MLE

$$P(w_i) = \frac{\#(w_i)}{|C_{uni}|} \quad P(h_i w_i) = \frac{\#(h_i w_i)}{|C_{n-gram}|}$$

If  $(h_i, m_i)$  is not observed in training corpus,  $P(w_i | h_i) = 0$

$P(\text{they, talk}) = P(\text{they} | \#) P(\text{talk} | \text{they}) = 0$

never observed (they talk) in training data

smoothing

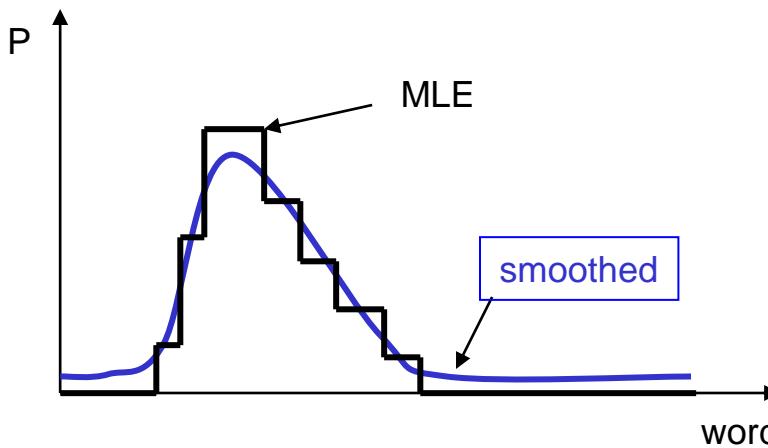
---



# Smoothing

---

Goal: assign a low probability to words or n-grams not observed in the training corpus





# Smoothing methods

---

n-gram:  $\alpha$

Change the freq. of occurrences

Laplace smoothing (add-one):

Good-Turing

change the freq.  $r$  to

$$P_{\text{add-one}}(\alpha | C) = \frac{|\alpha| + 1}{\sum_{\alpha_i \in V} (|\alpha_i| + 1)}$$

$$r^* = (r + 1) \frac{n_{r+1}}{n_r}$$



# Smoothing (cont' d)

---

Combine a model with a lower-order model  
Backoff (Katz)

Interpolation (Jelinek-Mercer)

$$P_{Katz}(w_i | w_{i-1}) = \begin{cases} P_{GT}(w_i | w_{i-1}) & \text{if } |w_{i-1}w_i| > 0 \\ \alpha(w_{i-1})P_{Katz}(w_i) & \text{otherwise} \end{cases}$$

$$P_{JM}(w_i | w_{i-1}) = \lambda_{w_{i-1}} P_{ML}(w_i | w_{i-1}) + (1 - \lambda_{w_{i-1}}) P_{JM}(w_i)$$

---



# Examples of utilization

---

Predict the next word

$$\operatorname{argmax}_w P(w \mid \text{previous words})$$

Used in input (predict the next letter/word on cellphone)

Use in machine aided human translation

Source sentence

Already translated part

Predict the next translation word or phrase

$$\operatorname{argmax}_w P(w \mid \text{previous trans. words, source sent.})$$



## Quality of a statistical language model

---

Test a trained model on a test collection

Try to predict each word

The more precisely a model can predict the words, the better is the model

Perplexity (the lower, the better)

Given  $P(w_i)$  and a test text of length N

Harmonic mean of probability

$$\frac{1}{N} \sum_{i=1}^N \frac{1}{P(w_i)}$$

At each word, how many choices does the model propose?

Perplexity=32 ~ 32 words could fit this position

---



## Sufficient training data

The longer is n (n-gram), the lower is perplexity

## Limited data

When n is too large, perplexity decreases

Data sparseness (sparsity)

In many NLP researches, one uses 5-grams or 6-grams

Google books n-gram (up to 5-

grams)<https://books.google.com/ngrams>



# More than predicting words

---

## Speech recognition

Training corpus = signals + words

probabilities:  $P(\text{signal}|\text{word})$ ,  $P(\text{word}_2|\text{word}_1)$

Utilization: signals      sequence of words

## Statistical tagging

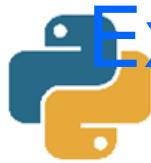


Training corpus = words + tags (n, v)

Probabilities:  $P(\text{word}|\text{tag})$ ,  $P(\text{tag}_2|\text{tag}_1)$

Utilization: sentence      sequence of tags





# Example of utilization

---

Speech recognition (simplified)

$$\begin{aligned} & \operatorname{argmax}_{w_1, \dots, w_n} P(w_1, \dots, w_n | s_1, \dots, s_n) \\ &= \operatorname{argmax}_{w_1, \dots, w_n} P(s_1, \dots, s_n | w_1, \dots, w_n) * P(w_1, \dots, \\ & w_n) \\ &= \operatorname{argmax}_{w_1, \dots, w_n} \prod_i P(s_i | w_1, \dots, w_n) * P(w_i | w_{i-1}) \\ &= \operatorname{argmax}_{w_1, \dots, w_n} \prod_i P(s_i | w_i) * P(w_i | w_{i-1}) \end{aligned}$$

Argmax - Viterbi search

probabilities:

$P(\text{signal} | \text{word}),$

$P(\text{***} | \text{ice-cream}) = P(\text{***} | \text{I scream}) = 0.8;$

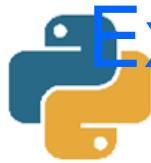
$P(\text{word2} | \text{word1})$

$P(\text{ice-cream} | \text{eat}) > P(\text{I scream} | \text{eat})$

Input speech signals  $s_1, s_2, \dots, s_n$

I eat ice-cream. > I eat I scream.

---



# Example of utilization

---

Statistical tagging

Training corpus = word + tag (e.g. Penn Tree Bank)

For  $w_1, \dots, w_n$ :

$$\operatorname{argmax}_{\text{tag}_1, \dots, \text{tag}_n} \prod_i P(w_i | \text{tag}_i) * P(\text{tag}_i | \text{tag}_{i-1})$$

probabilities:

$$P(\text{word} | \text{tag})$$

$$P(\text{change} | \text{noun}) = 0.01, P(\text{change} | \text{verb}) = 0.015;$$

$$P(\text{tag}_2 | \text{tag}_1)$$

$$P(\text{noun} | \text{det}) >> P(\text{verb} | \text{det})$$

Input words:  $w_1, \dots, w_n$

I give him the change.

pronoun verb pronoun det noun >

pronoun verb pronoun det verb



## Some improvements of the model

---

Class model

Instead of estimating  $P(w_2|w_1)$ , estimate  $P(w_2|\text{Class1})$

$P(\text{me}|\text{take})$  v.s.  $P(\text{me}|\text{Verb})$

More general model

Less data sparseness problem

Skip model

Instead of  $P(w_i|w_{i-1})$ , allow  $P(w_i|w_{i-k})$

Allow to consider longer dependence



# State of the art on POS-tagging

---

POS = Part of speech (syntactic category)

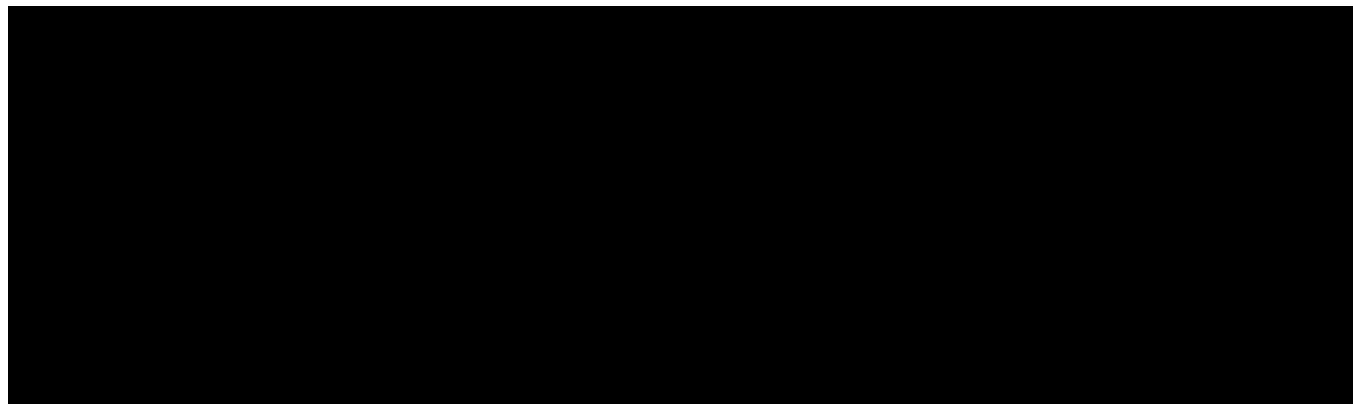
Statistical methods

Training based on annotated corpus (text with tags annotated manually)

Penn Treebank: a set of texts with manual annotations

<http://www.cis.upenn.edu/~treebank/>

---



One can learn:

$$P(w_i)$$

$$P(\text{Tag} \mid w_i), P(w_i \mid \text{Tag})$$

$$P(\text{Tag}_2 \mid \text{Tag}_1), P(\text{Tag}_3 \mid \text{Tag}_1, \text{Tag}_2)$$

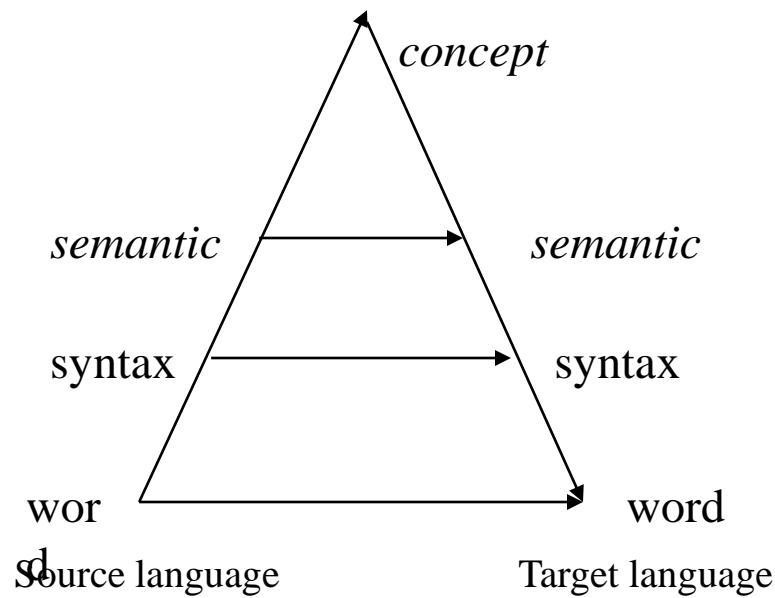
...



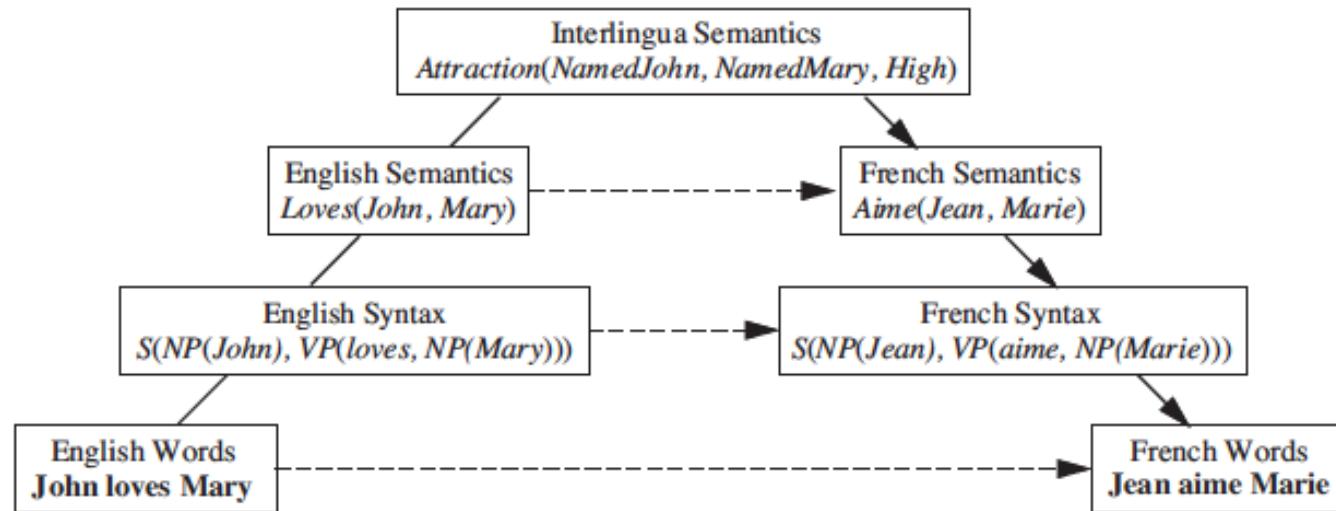
# State of the art of MT

---

Vauquois triangle (simplified)



# Triangle of Vauguois





# State of the art of MT (cont'd)

---

General approach:

Word / term: dictionary

Phrase

Syntax

Limited “semantics” to solve common ambiguities

Typical example: Systran

---



## Word/term level

---

Choose one translation word

Sometimes, use context to guide the selection of translation words

The boy grows: grandir

... grow potatoes: cultiver



---

Pomme de terre -> potatoe

Find a needle in haystacks ->大海捞针

---



# Statistical machine translation

---

$$\begin{aligned}\operatorname{argmax}_F P(F|E) &= \operatorname{argmax}_F P(E|F) P(F) / P(E) \\ &= \operatorname{argmax}_F P(E|F) P(F)\end{aligned}$$

$P(E|F)$ : translation model

$P(F)$ : language model, e.g. trigram model

More to come later on translation model

---



---

Traditional NLP approaches: symbolic, grammar, ...

More recent approaches: statistical

For some applications: statistical approaches are better (tagging, speech recognition, ...)

For some others, traditional approaches are better (MT)

Trend: combine statistics with rules (grammar)

E.g.

Probabilistic Context Free Grammar (PCFG)

Consider some grammatical connections in statistical approaches

NLP still a very difficult problem

---



# Image Segmentation using Differential Evolution

---



## **Mentor Information**

---

### ***Faculty Mentor Info:***

*Name: Michael Georgopoulos  
Email: michaelg@ucf.edu  
Phone: 407-312-0957*

### ***Graduate Student Mentor Info:***

*Name: Tiantian Zhang  
Email: watera427@gmail.com  
Phone: 407-284-0799*



---

## Motivation

- ❑ Image segmentation is based on non-supervised pattern classification techniques. In image segmentation, different features are extracted and grouped into different clusters, and thus the image is more visually preferable to reading useful information of differently grouped semantic regions.
  
- ❑ Image segmentation has been widely used in various fields [1].
  - Computer-aided diagnosis (CAD) for cancer detection,
  - Remote sensing image analysis,
  - Outdoor object recognition & robot vision,
  - Content-based image and marketplace decision support.

[1] Jose Alfredo F. Costa and Jackson G. de Souza, "Image Segmentation through Clustering Based on Natural Computing Techniques", pp. 57-82, In: Image Segmentation, Pei-Gee Ho (Ed.), 2011

---

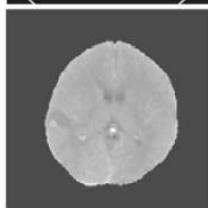
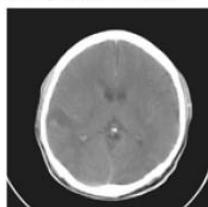


---

## Applications of Image Segmentation

### □ Medical Image Analysis

CT Head Image & Its  
Intracranial Area:



Method:

Abnormal Regions:

Brain Matter:

CSF:

k-means

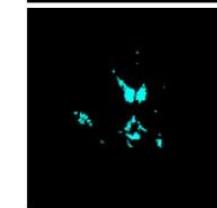
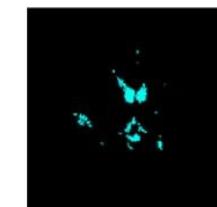
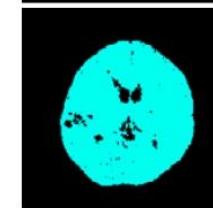
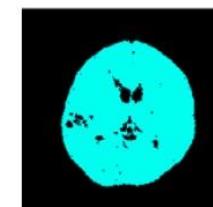


Figure from [2]

[2]Tong Hau Lee, Mohammad Faizal Ahmad Fauzi, Ryoichi Komiya, "Segmentation of CT Brain Images Using Unsupervised Clusterings", Journal of Visualization, Vol. 12, No. 2 (2009) 131-138 , 2009



## Applications of Image Segmentation

### ❑ Remote Sensing Image Analysis

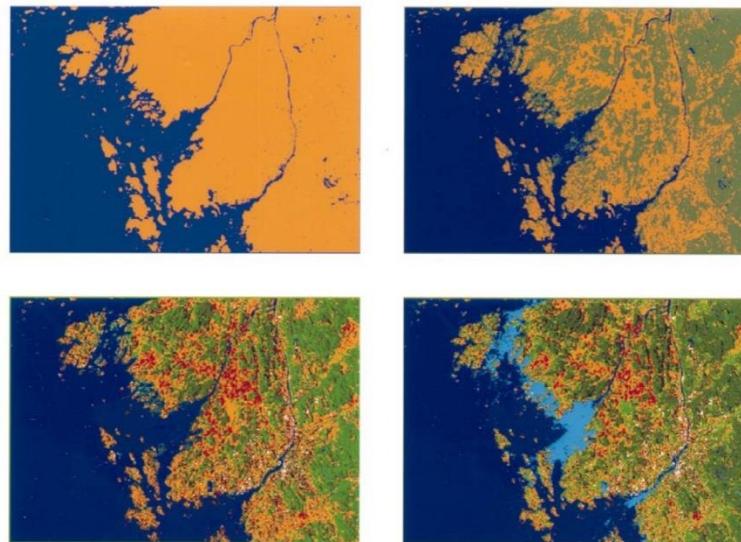


Figure from [3]

Fig. 7. Four clustering results for Gothenburg, number of clusters are two (top left), three (top right), nine (bottom left), and ten (bottom right). Shades of blue: water; shades of green: forests; white: man-made structures (urban); orange: open areas (other); red: farmlands; and black: system effects (foreshortening and layover). The light blue is the sea ice from the winter images.

[3] Patrik B. G. Dammert, Jan I. H. Askne, Sharon Kuhlmann, “Unsupervised Segmentation of Multitemporal Interferometric SAR Images”, IEEE Transactions on Geoscience and Remote Sensing, vol. 37, no. 5, Sep. 1999



## Applications of Image Segmentation

- Document Analysis – Background removal

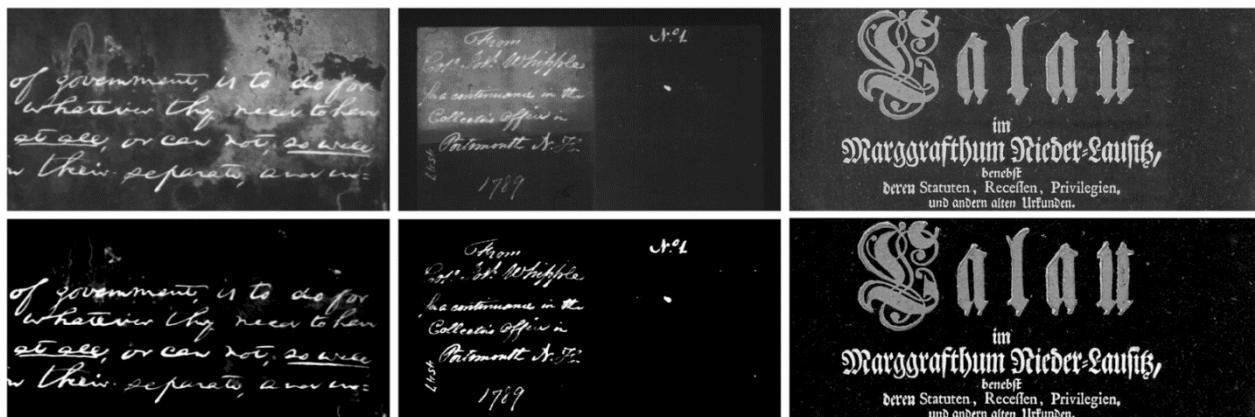


Fig. 11. Background removal based on h-component tree processing. (Top) Original document images and (bottom) corresponding results of the proposed background removal method.

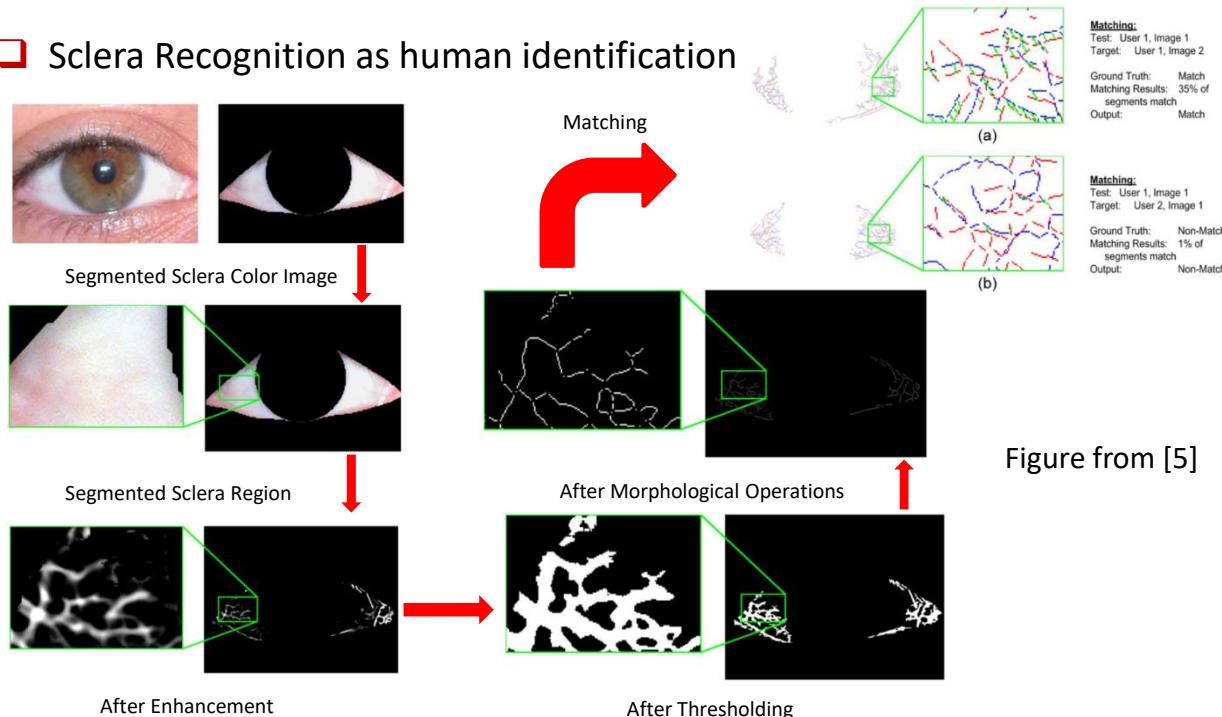
Figure from [4]

[4] Benjamin Perret, Sébastien Lefèvre, Christophe Collet, and Éric Slezak, "Hyperconnections and Hierarchical Representations for Grayscale and Multiband Image Processing", IEEE Transactions on Image Processing, Vol. 21, No. 1, January 2012



## Applications of Image Segmentation

### ❑ Sclera Recognition as human identification



[5] Zhi Zhou, Eliza Yingzi Du, N. Luke Thomas, Edward J. Delp, "A New Human Identification Method: Sclera Recognition", IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans, Vol. 42, No. 3, May 2012



---

## ***Proposed Project Work***

- ❑ In this project, we are going to do Image Clustering using a variety of clustering approaches.
  
- ❑ Typical clustering algorithms are:
  - K-means Algorithms
  - ISODATA Clustering Algorithms
  - Fuzzy C-Means (FCM)
  - Expectation-Maximization (EM)
  - K-harmonic Means (KHM)
  - Artificial Neural Networks (ANN)



---

## Proposed Project Work

- ❑ The disadvantages of these algorithms are [6]:
  - The performance of some algorithms is highly dependent on initial conditions;
  - The number of clusters must be known a-priori information for some of algorithms;
  - Clustering is deemed to be a particular kind of NP-hard problem. Therefore, local search algorithms are inefficient sometimes.
- ❑ Evolutionary Algorithms are designed to be population-based global search algorithms. They are metaheuristics widely believed to be effective on NP-hard problems, being able to provide near-optimal solutions to such problems in reasonable time [6].
- ❑ Differential Evolution is a simple and robust optimizer compared with other EAs.

[6] Eduardo R. Hruschka, Ricardo J. G. B. Campello, Alex A. Freitas, André C. P. L. F. de Carvalho, "A Survey of Evolutionary Algorithms for Clustering", IEEE Transactions on Systems, Man, and Cybernetics, 2012

---



---

## Proposed Project Work

❑ Advantages of DE [7]:

- Compared to most EAs, DE is much simpler and thus straightforward to implement.
- Despite its simplicity, DE exhibits much better performance in comparison with several other EAs.
  - DE finished third in the First International Contest on Evolutionary Optimization in May 1996 (CEC, 1996).
  - DE turned out to be the best evolutionary algorithm for solving the real-valued test function suite of the 1<sup>st</sup> ICEO
  - In the 2005 CEC competition on real time parameter optimization, on 10-D problems classical DE secured 2nd rank, while a self-adaptive DE variant called SaDE secured third rank.
- The number of control parameters in DE are very few (Cr, F and Np in classical DE)

❑ DE has been widely used in signal and image processing, pattern recognition, chemical engineering, electrical power system, and bioinformatics, etc. [7]

[7] Swagatam Das, Ponnuthurai Nagaratnam Suganthan, Differential Evolution: A Survey of the State-of-the-Art, IEEE Transactions on Evolutionary Computation, Vol. 15, pp. 4-31, Feb. 2011.

---



---

## ***Proposed Work; Related Knowledge***

**❑ Image Fundamentals:**

- Image Representation, Pixel Intensity, Color, Gray-Scale Images
- Useful Image Transformations

**❑ Mathematical Knowledge:**

- Vectors and Operations on Vectors
- Geometrical Representation of Vectors
- Euclidean Distance calculation in multi-dimensional space.

**❑ Programming Techniques:**

- Matlab Fundamentals
- Image Processing Toolbox Functionality

**❑ Machine Learning Knowledge:**

- K-Means Clustering Algorithm
  - Differential Evolution and variant Algorithms
-



---

## ***Timeline for Proposed Work***

- ❑ Week 1:
  - How to use MATLAB
- ❑ Week 2:
  - Fundamentals of Image Processing
  - Introduction to Clustering
  - K-means Algorithm and its implementation
- ❑ Week 3:
  - K-means algorithm and image clustering applications
  - DE algorithm and its variants
- ❑ Week 4:
  - DE implementation and optimization problem applications
  - ACDE algorithm
- ❑ Week 5:
  - Implementation of ACDE and image clustering applications
- ❑ Week 6:
  - Work on deliverables



---

## References

- [1] Jose Alfredo F. Costa and Jackson G. de Souza, "Image Segmentation through Clustering Based on Natural Computing Techniques", pp. 57-82, In: Image Segmentation, Pei-Gee Ho (Ed.), 2011
  - [2] Tong Hau Lee, Mohammad Faizal Ahmad Fauzi, Ryoichi Komiya, "Segmentation of CT Brain Images Using Unsupervised Clusterings", Journal of Visualization, Vol. 12, No. 2 (2009) 131-138 , 2009
  - [3] Patrik B. G. Dammert, Jan I. H. Askne, Sharon Kuhlmann, "Unsupervised Segmentation of Multitemporal Interferometric SAR Images", IEEE Transactions on Geoscience and Remote Sensing, vol. 37, no. 5, Sep. 1999
  - [4] Benjamin Perret, Sébastien Lefèvre, Christophe Collet, and Éric Slezak, "Hyperconnections and Hierarchical Representations for Grayscale and Multiband Image Processing", IEEE Transactions on Image Processing, Vol. 21, No. 1, January 2012
  - [5] Zhi Zhou, Eliza Yingzi Du, N. Luke Thomas, Edward J. Delp, "A New Human Identification Method: Sclera Recognition", IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans, Vol. 42, No. 3, May 2012
  - [6] Eduardo R. Hruschka, Ricardo J. G. B. Campello, Alex A. Freitas, André C. P. L. F. de Carvalho, "A Survey of Evolutionary Algorithms for Clustering", IEEE Transactions on Systems, Man, and Cybernetics, 2012
  - [7] Swagatam Das, Ponnuthurai Nagaratnam Suganthan, Differential Evolution: A Survey of the State-of-the-Art, IEEE Transactions on Evolutionary Computation, Vol. 15, pp. 4-31, Feb. 2011.
-