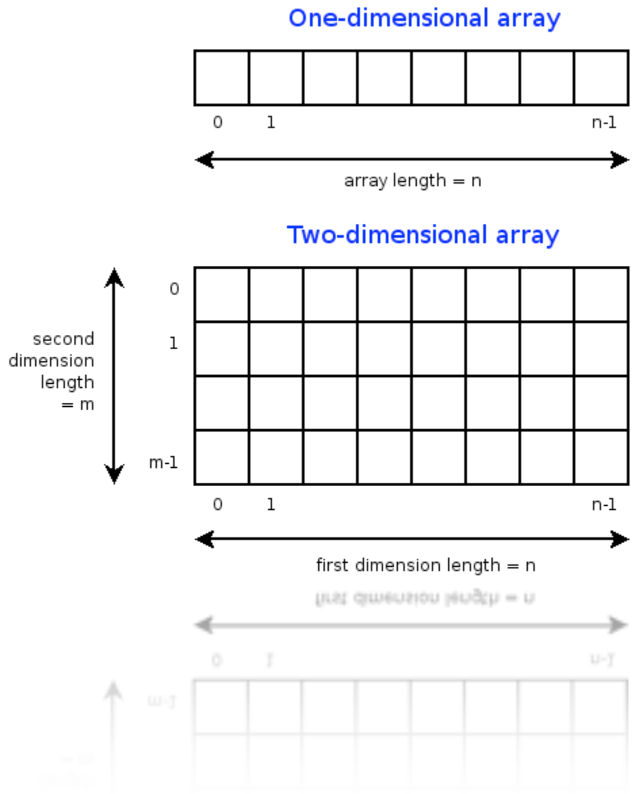
Arrays

The array data type is a variable that can be indexed. Typically, you specify the type of the stored objects and its size at creation time.



Examples

```
>>> np.array([1, 2, 3])
array([1, 2, 3])
```

Upcasting:

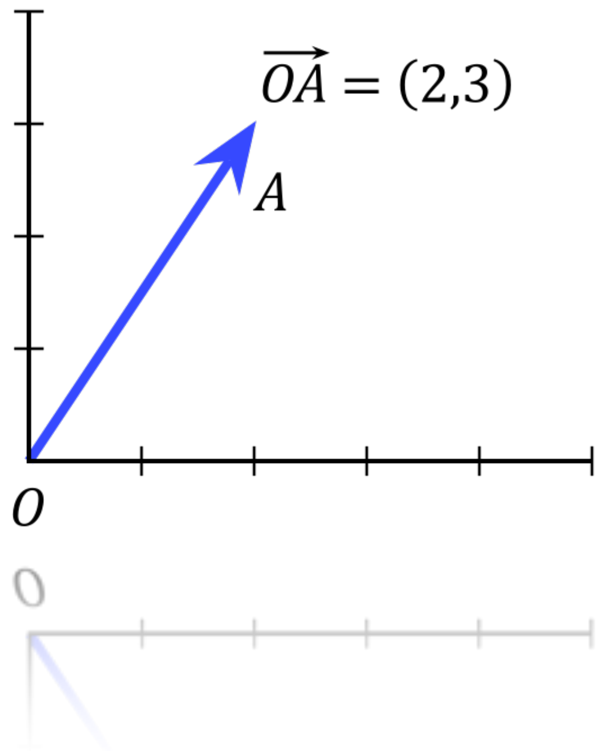
```
>>> np.array([1, 2, 3.0])
array([ 1.,  2.,  3.])
```

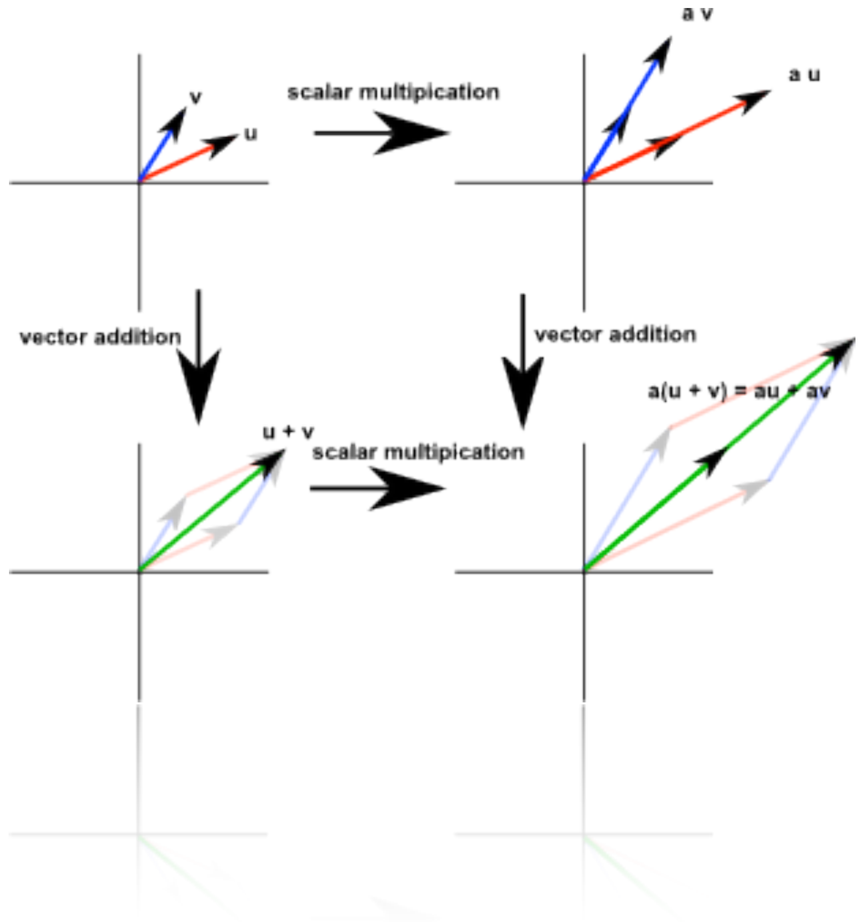
More than one dimension:

```
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

Vectors

Vectors are one-dimensional array. Feature vectors in ML are representations of data points.





v
u
scalar multipication
a v
a u
vector addition
vector addition
u + v
scalar multipication
a(u + v) = au + av

Matrices

Matrices are rectangular arrays of numbers.

$$
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
$$

In two dimensions every rotation matrix has the following form:

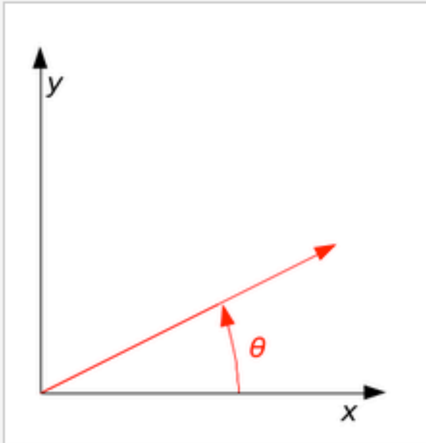
$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

This rotates column vectors by means of the following matrix multiplication:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

So the coordinates (x',y') of the point (x,y) after rotation are:

$$x' = x\cos\theta - y\sin\theta,$$
$$y' = x\sin\theta + y\cos\theta.$$



A counterclockwise rotation of a vector through angle θ. The vector is initially aligned with the x-axis.

The direction of vector rotation is counterclockwise if θ is positive (e.g. 90°), and clockwise if θ is negative (e.g. -90°). Thus the clockwise rotation matrix is found as:

$$R(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

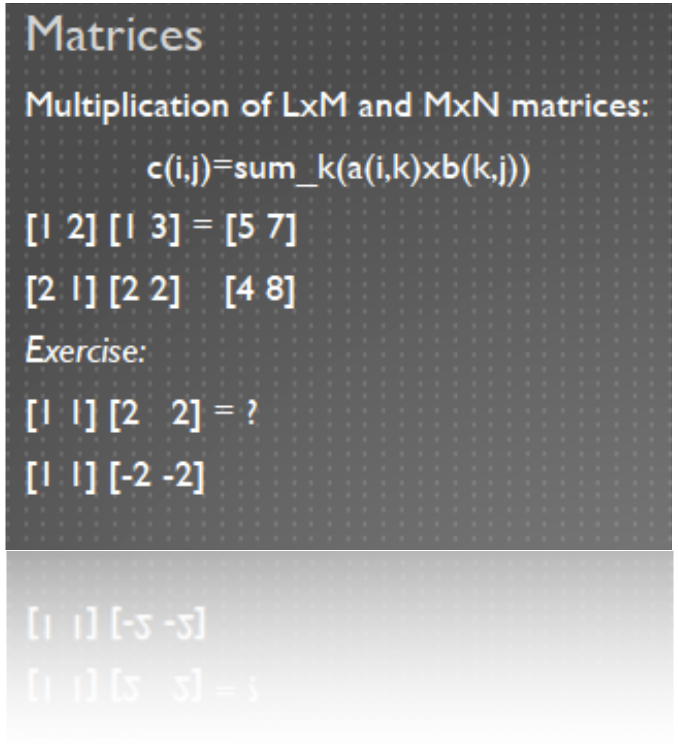
Note that the two-dimensional case is the only non-trivial (e.g. one dimension) case where the rotation matrices group is commutative, so that it does not matter the order in which multiple rotations are performed.

```
>>> measurements = [
...     {'city': 'Dubai', 'temperature': 33.},
...     {'city': 'London', 'temperature': 12.},
...     {'city': 'San Fransisco', 'temperature': 18.},
... ]

>>> from sklearn.feature_extraction import DictVectorizer
>>> vec = DictVectorizer()

>>> vec.fit_transform(measurements).toarray()
array([[  1.,   0.,   0.,  33.],
       [  0.,   1.,   0.,  12.],
       [  0.,   0.,   1.,  18.]])

>>> vec.get_feature_names()
['city=Dubai', 'city=London', 'city=San Fransisco', 'temperature']
```

$$
\begin{aligned}
A &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix} \\
B &= \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \vdots & \vdots & & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nk} \end{bmatrix} \\
A + B &= \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1k} + b_{1k} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2k} + b_{2k} \\ \vdots & \vdots & & \vdots \\ a_{n1} + b_{n1} & a_{n2} + b_{n2} & \cdots & a_{nk} + b_{nk} \end{bmatrix} \\
cA &= \begin{bmatrix} ca_{11} & ca_{12} & \cdots & ca_{1k} \\ ca_{21} & ca_{22} & \cdots & ca_{2k} \\ \vdots & \vdots & & \vdots \\ ca_{n1} & ca_{n2} & \cdots & ca_{nk} \end{bmatrix}
\end{aligned}
$$

Computing with matrices

a) Transpose: A[i,j]=A.T[j,i]
b) Matrix multiplication
c) Pointwise multiplication



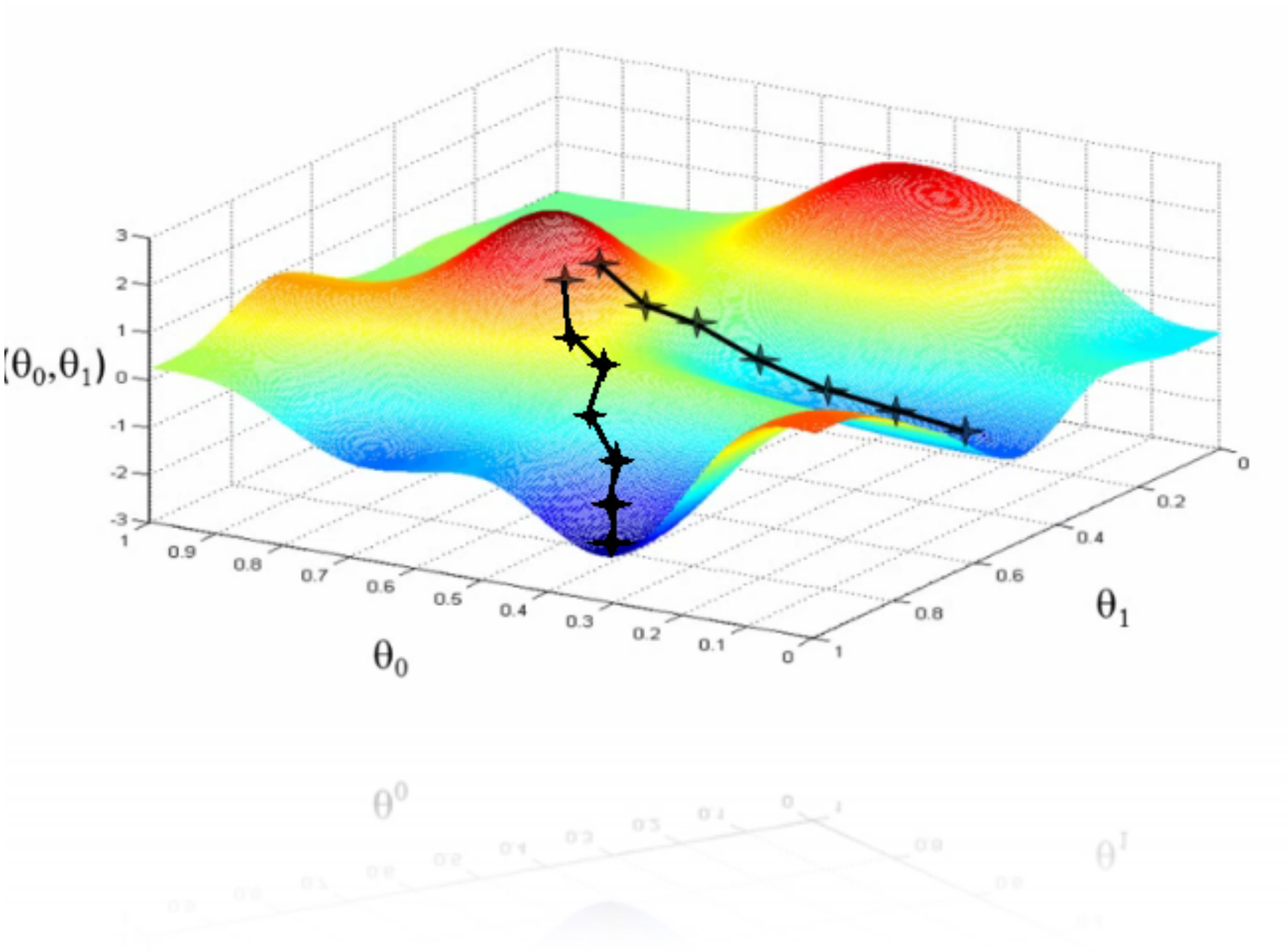
```
In [9]: a=np.array([0,1,2])

In [10]: b=np.array([1,2,1])

In [11]: a*b
Out[11]: array([0, 2, 2])

In [12]: dot(a,b)
Out[12]: 4
```

```
In [14]: a=np.array([[0,1,2],[0,1,2]])

In [15]: b=np.array([[1,2,1],[1,2,1]])

In [16]: a*b
Out[16]:
array([[0, 2, 2],
       [0, 2, 2]])

In [17]: dot(a,b)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/Library/Frameworks/Python.framework/Versions/7.2/Resources/<ipython-input-17-154c166d19c0> in
<module>()
----> 1 dot(a,b)

ValueError: objects are not aligned
```

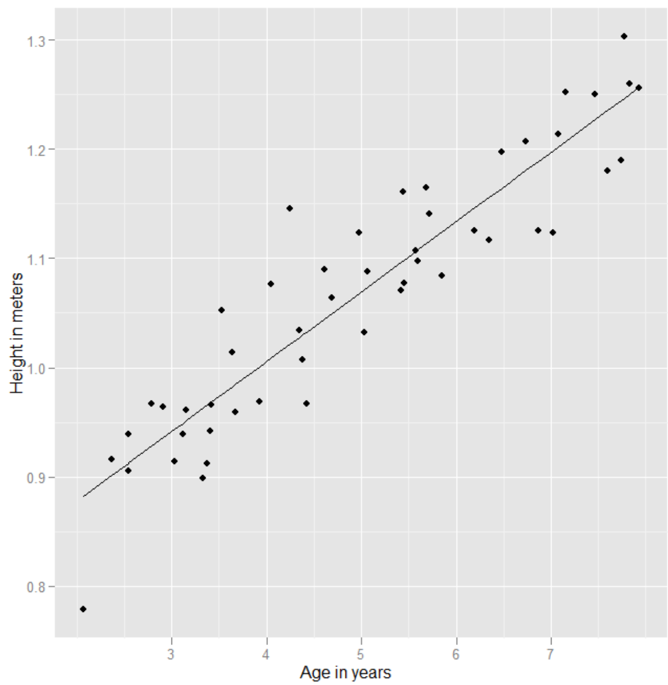
```
In [14]: a=np.array([[0,1,2],[0,1,2]])

In [18]: b=np.array([[1,2,1],[1,2,1],[1,2,1]])

In [19]: dot(a,b)
Out[19]:
```

Exercise!

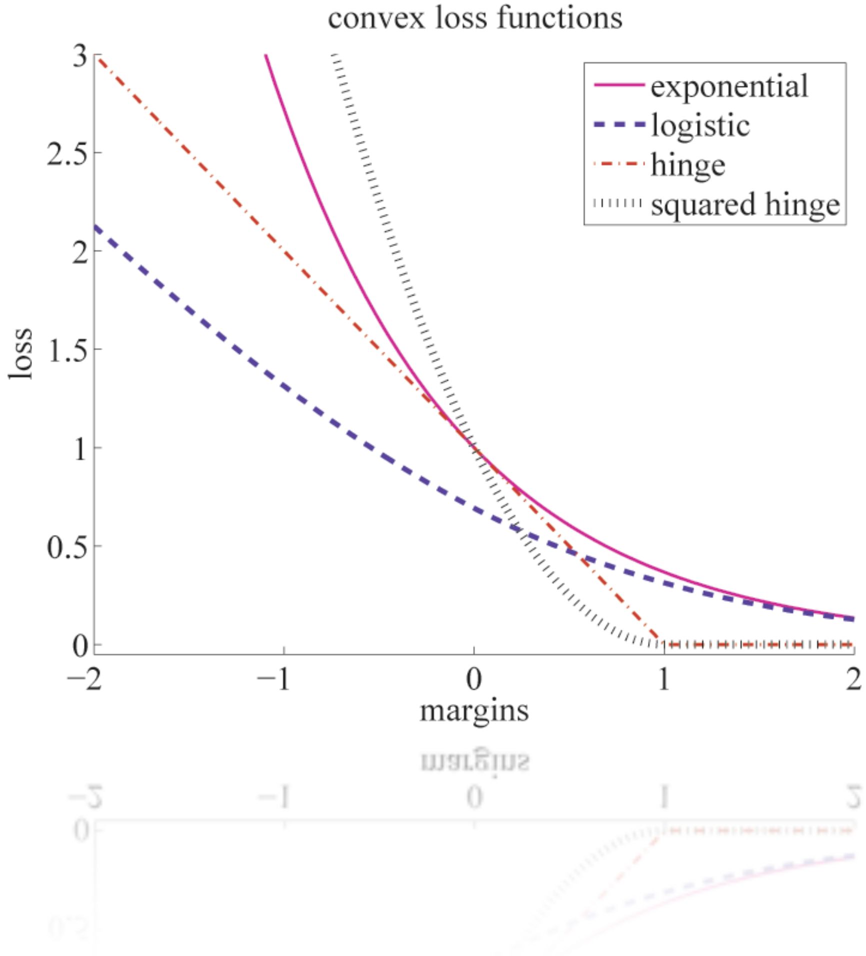
```
In [14]: a=np.array([[0,1,2],[0,1,2]])

In [18]: b=np.array([[1,2,1],[1,2,1],[1,2,1]])

In [19]: dot(a,b)
Out[19]:
array([[3, 6, 3],
       [3, 6, 3]])
```



Gradient descent



Linear regression


convex loss functions
loss
margins
3
2.5
2
1.5
1
0.5
0
−2
−1
0
1
2
exponential
logistic
hinge
squared hinge

Mini-batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

}

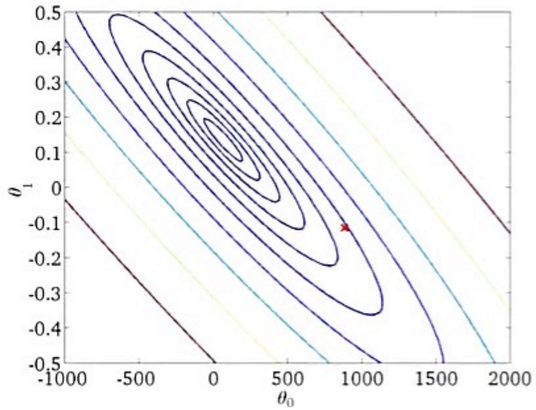
}

Andrew Ng

Stochastic gradient descent

1. Randomly shuffle (reorder) training examples

2. Repeat {
   for $i := 1, \ldots, m$ {
   $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$
   (for every $j = 0, \ldots, n$)
   }
   }



Andrew Ng