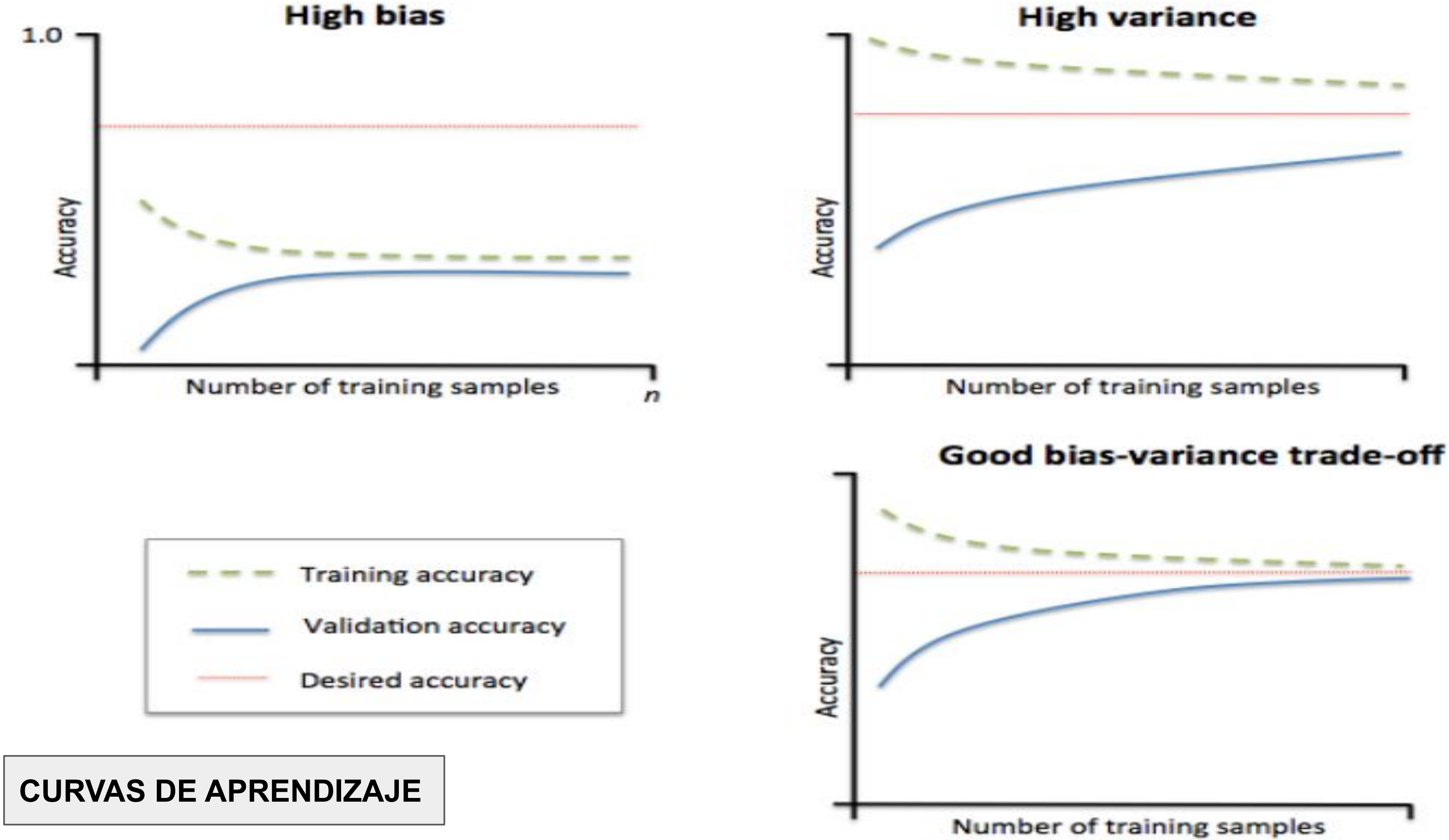
# APRENDIZAJE AUTOMÁTICO A GRAN ESCALA

★ **Descenso de gradientes con grandes conjuntos de datos**

# APRENDER CON GRANDES CONJUNTOS DE DATOS

Si tenemos una cantidad de datos muy grande, calcular el descenso de la gradiente puede ser costoso computacionalmente.

Entonces… **¿Cómo medir si un conjunto pequeño de datos puede hacer igual de bien su trabajo?**




## CURVAS DE APRENDIZAJE

# DESCENSO DE GRADIENTE ESTOCÁSTICO

**Batch gradient descent**

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

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

}

**TOMA TODOS LOS EJEMPLOS DE ENTRENAMIENTO PARA CADA INTERACCIÓN**

**VS**

**Stochastic gradient descent**

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

**TOMA UN SOLO EJEMPLO DE ENTRENAMIENTO**

# Pero realmente cual es el procedimiento…?

### Stochastic gradient descent

$$cost(\theta,(x^{(i)},y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)})-y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta,(x^{(i)},y^{(i)}))$$

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

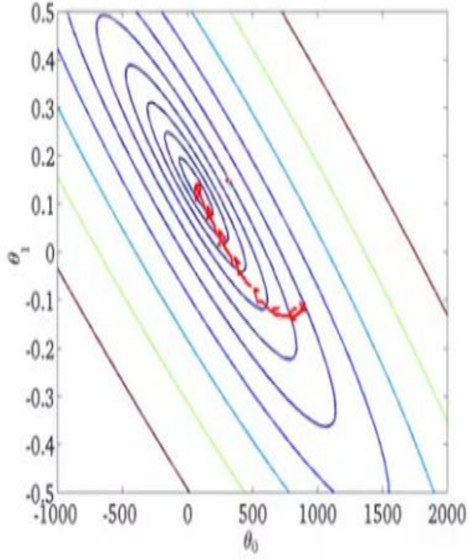
$\frac{\partial}{\partial\theta_j} cost(\theta,(x^{(i)},y^{(i)}))$

$(x^{(1)},y^{(1)}), (x^{(2)},y^{(2)}), (x^{(3)},y^{(3)}), \ldots$

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 $j = 0,\ldots,n$)
   }
   }



# DESCENSO DE GRADIENTE DE MINI-LOTES

Se diferencia del descenso de gradientes por lotes y estocástico en la cantidad de ejemplos de entrenamiento que toma para cada interacción. Toma ‘ b ’ cantidades.

b: Parámetro denominado ‘tamaño del mini-lote’

Puede algunas veces trabajar más rápido que el descenso de gradiente estocástico.

# ¿Cómo funciona?

## 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 $j = 0, \ldots, n$)

}

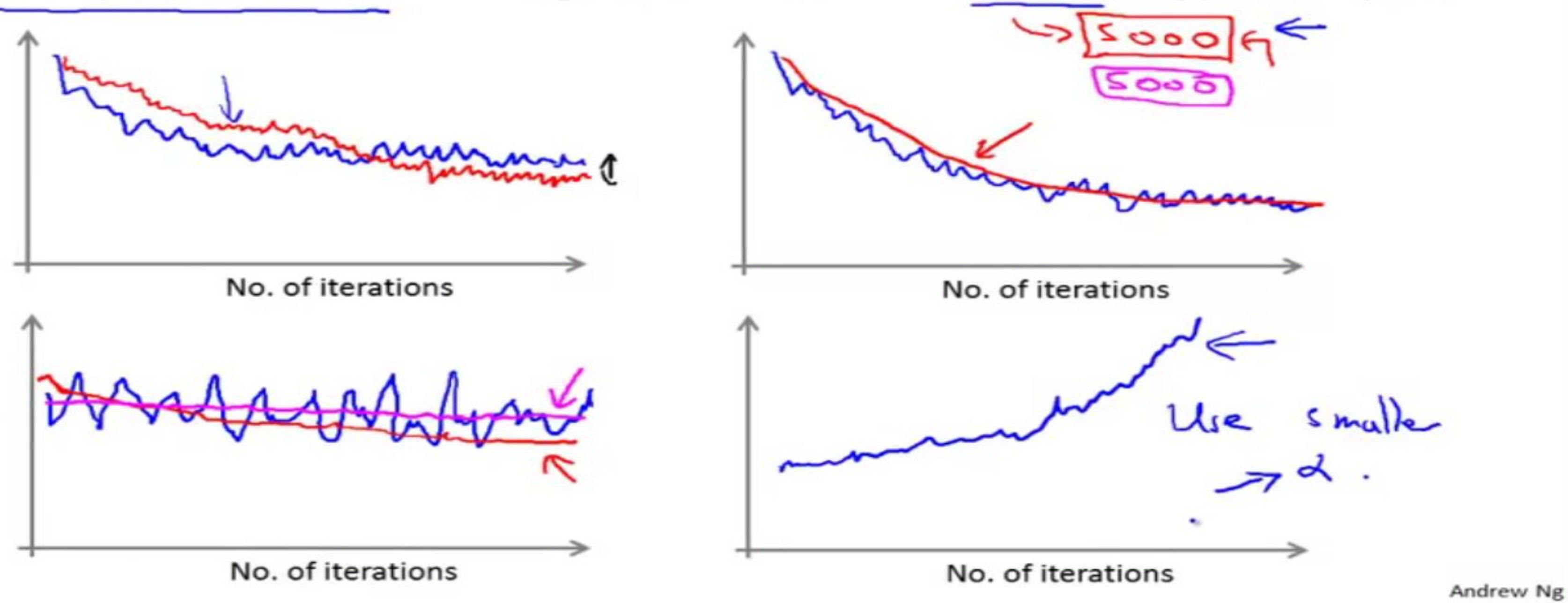
}

Teniendo un buen parámetro b y una vectorizaciòn adecuada , Mini-batch gradient descent es más eficiente que el descenso de gradiente por lotes.

# CONVERGENCIA DE DESCENSO DE GRADIENTE ESTOCÁSTICO

**Checking for convergence**

Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples



# Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset.
2. Repeat {

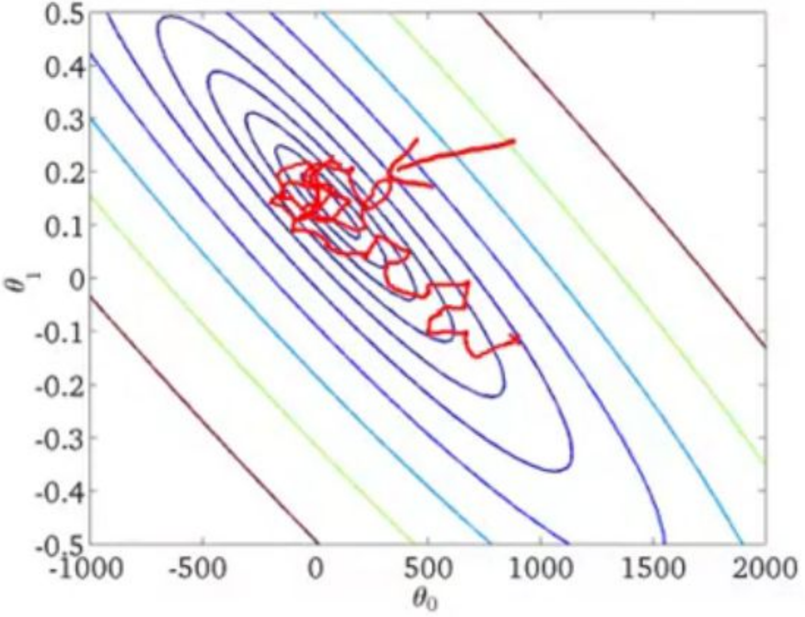
    for $i := 1, \ldots, m$ {

    $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

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

    }

}




Learning rate $\alpha$ is typically held constant. Can slowly decrease $\alpha$ over time if we want $\theta$ to converge. (E.g. $\alpha = \frac{\texttt{const1}}{\texttt{iterationNumber + const2}}$ )

Andrew Ng

# Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset.
2. Repeat {

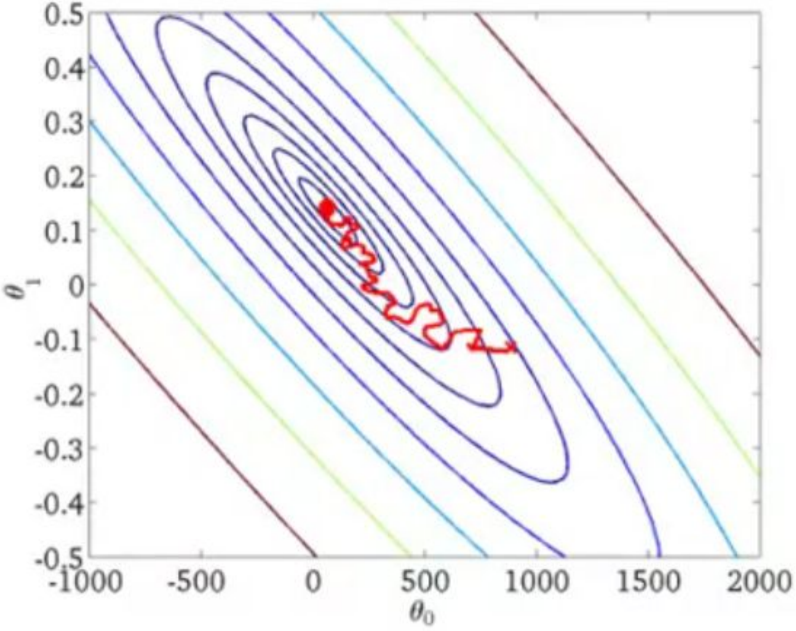
    for $i := 1, \ldots, m$ {

    $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

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

    }

}




Learning rate $\alpha$ is typically held constant. Can slowly decrease $\alpha$ over time if we want $\theta$ to converge. (E.g. $\alpha = \frac{\texttt{const1}}{\boxed{\texttt{iterationNumber}} + \texttt{const2}}$ )

# TEMAS AVANZADOS

- ENTORNO DE APRENDIZAJE EN LÍNEA:

  Una de sus principales ventajas es que se puede adaptar a las preferencias de los usuarios y con un continuo flujo de datos se puede hallar una buena predicciòn que los usuarios. Tenemos un ejemplo como el CTR.

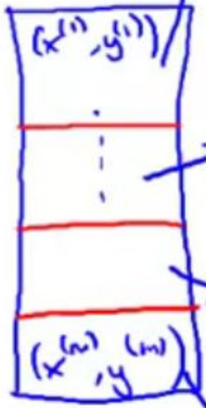
- REDUCCIÓN DE MAPAS Y PARALELISMO DE DATOS:

  Se emplea el enfoque MAP-REDUCE con datos muy grandes , los cuales son computacionalmente muy costosos realizar en una sola máquina.

$m = 400$ $\qquad$ $m = 400{,}000{,}000$

**Map-reduce**

Batch gradient descent: $\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{\boxed{400}} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$




Machine 1: Use $(x^{(1)}, y^{(1)}), \ldots, (x^{(100)}, y^{(100)})$.

$$temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Machine 2: Use $(x^{(101)}, y^{(101)}), \ldots, (x^{(200)}, y^{(200)})$.

$$temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Machine 3: Use $(x^{(201)}, y^{(201)}), \ldots, (x^{(300)}, y^{(300)})$.

$$temp_j^{(3)} = \sum_{i=201}^{300} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Machine 4: Use $(x^{(301)}, y^{(301)}), \ldots, (x^{(400)}, y^{(400)})$.

$$temp_j^{(4)} = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

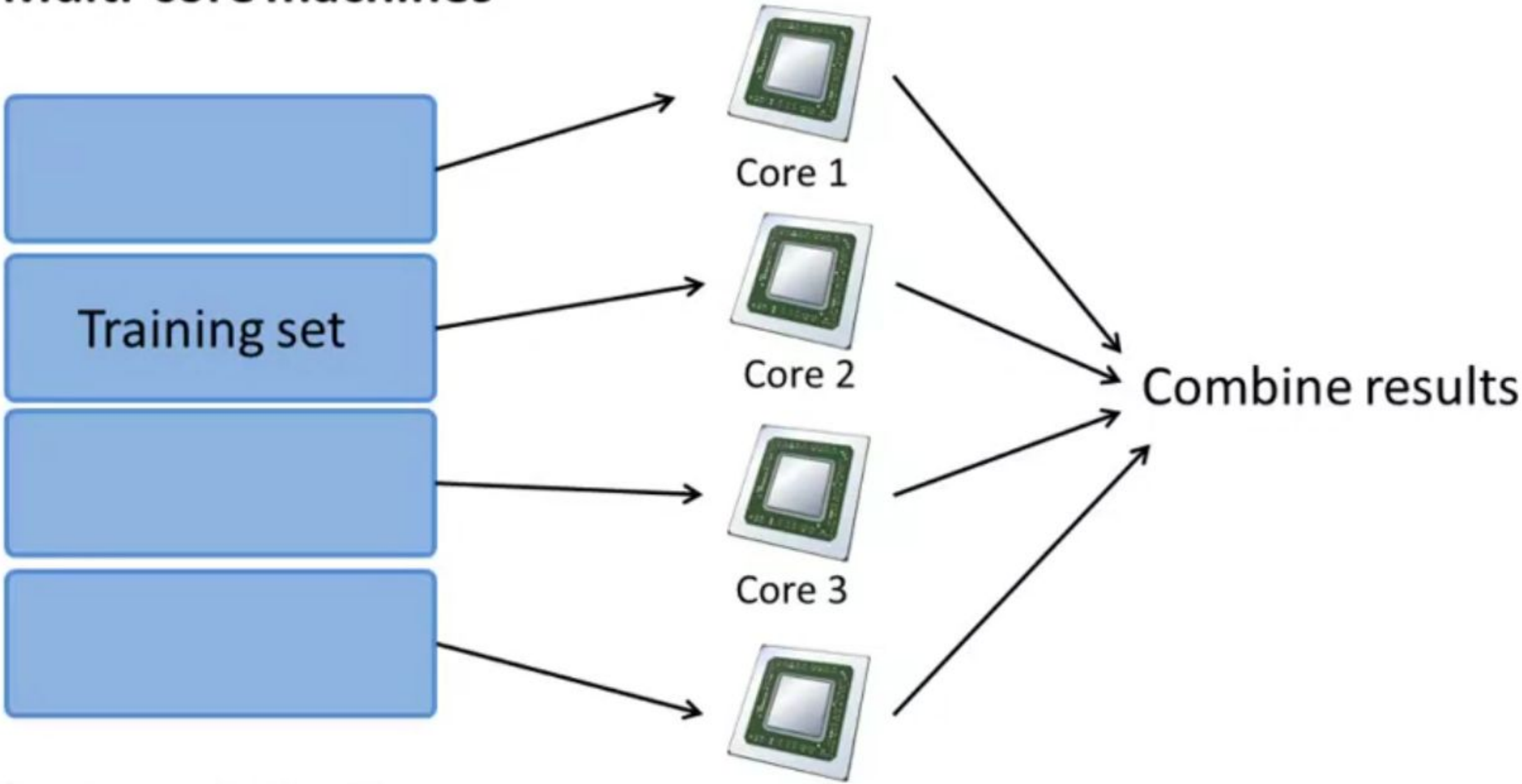
Combine:

$$\theta_j := \theta_j - \alpha \frac{1}{400} \left( temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)} \right)$$

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

[Jeffrey Dean and Sanjay Ghemawat]

# Multi-core machines




[http://openclipart.org/detail/100267/cpu-(central-processing-unit)-by-ivak-100267]