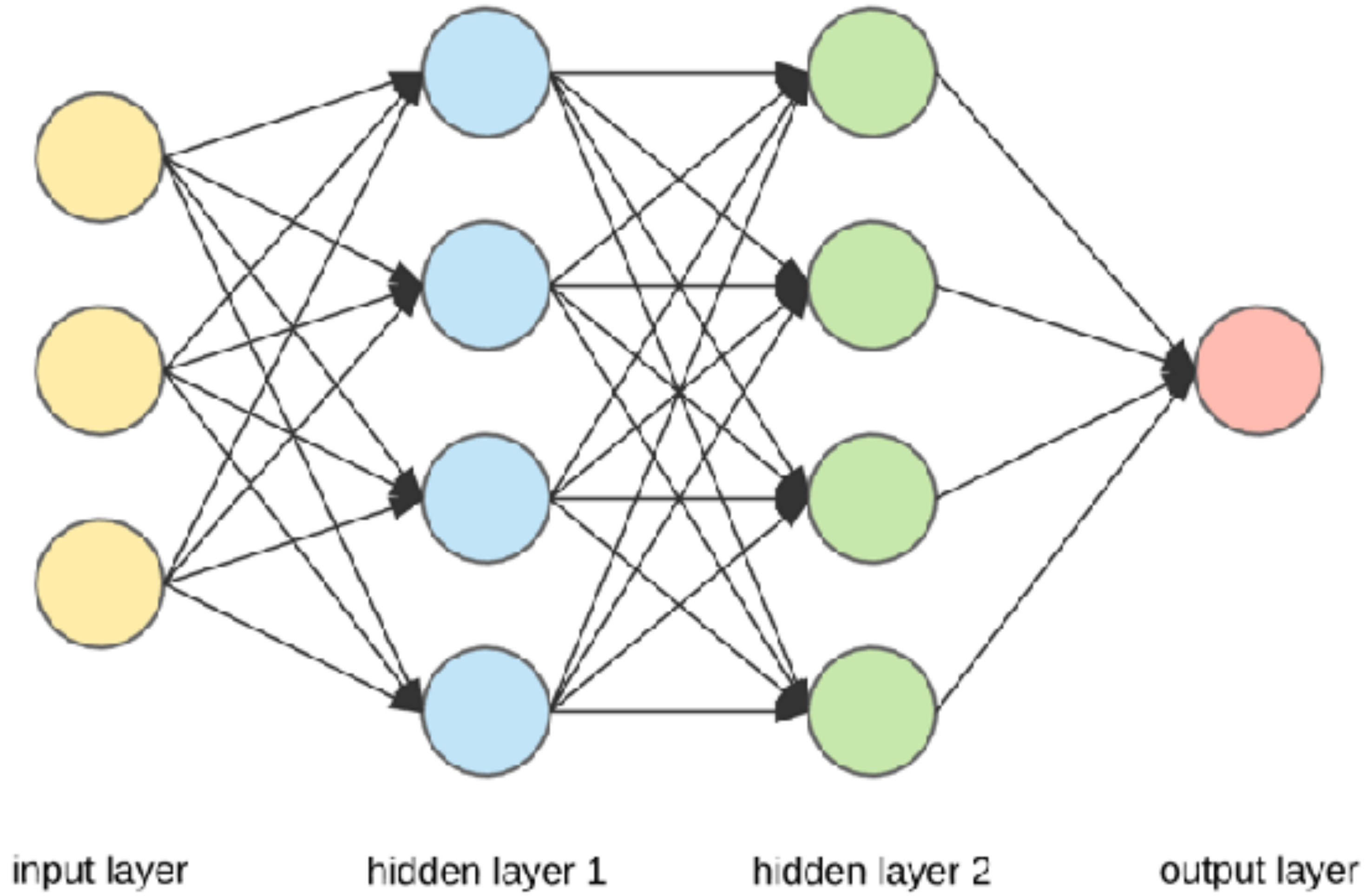
NN & Optimization

Yue Wu

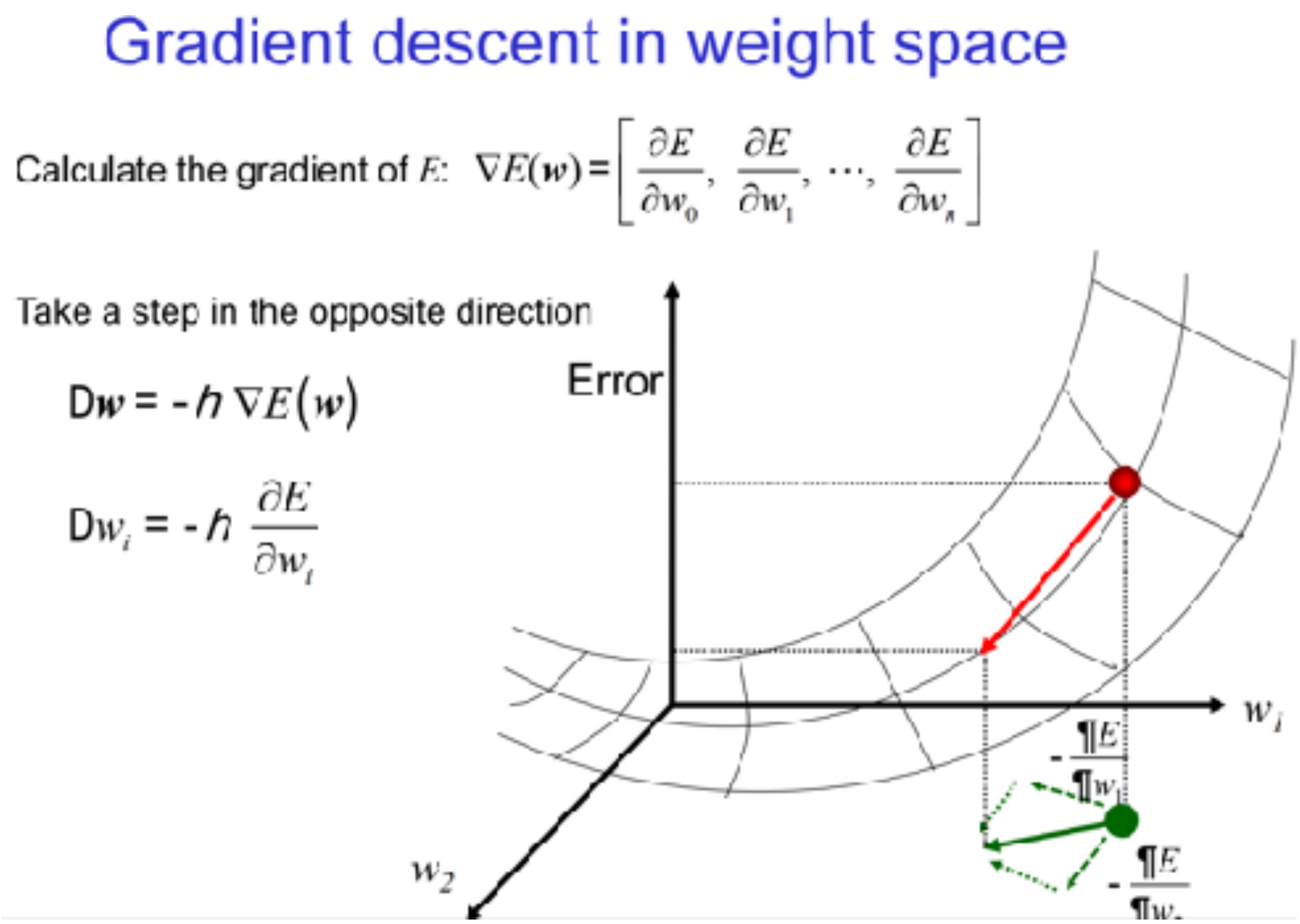
Questions?



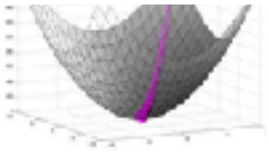
Back Prop (Isn’t This Trivial?)

- When I first understood what backpropagation was, my reaction was: “Oh, that’s just the chain rule! How did it take us so long to figure out?” I’m not the only one who’s had that reaction. It’s true that if you ask “is there a smart way to calculate derivatives in feedforward neural networks?” the answer isn’t that difficult.
- But I think it was much more difficult than it might seem. You see, at the time backpropagation was invented, people weren’t very focused on the feedforward neural networks that we study. It also wasn’t obvious that derivatives were the right way to train them. Those are only obvious once you realize you can quickly calculate derivatives. There was a circular dependency.

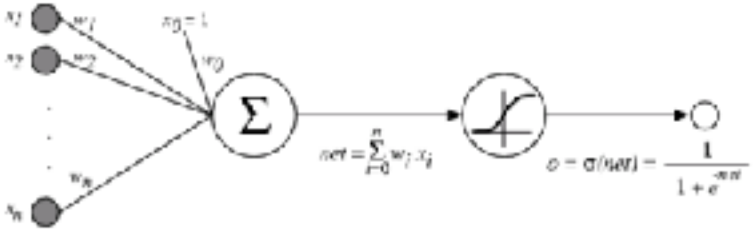
Gradient Descent



Backprop for Sigmoid



Given $(x_d, t_d)_{d \in D}$ find **w** to minimize $\sum_{d \in D} (o_d - t_d)^2$



o_d = observed unit output for x_d

$o_d = \sigma(net_d); \quad net_d = \sum_i w_i x_{i,d}$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

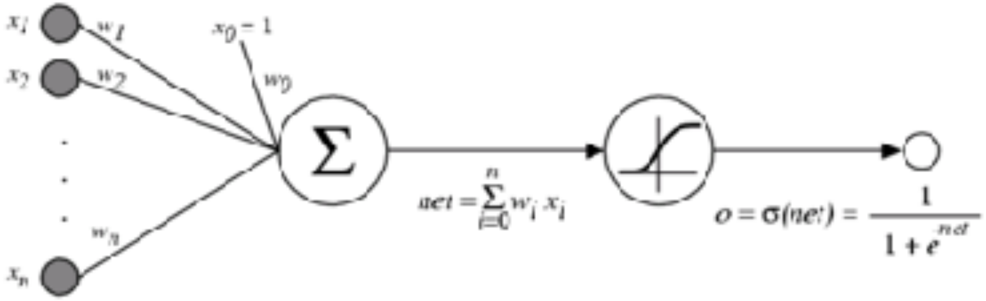
$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) = \sum_{d \in D} (t_d - o_d) \left(- \frac{\partial o_d}{\partial w_i} \right)$$

$$= - \sum_{d \in D} (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}$$

But we know: $\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$ and $\frac{\partial net_d}{\partial w_i} = \frac{\partial (\mathbf{w} \cdot \mathbf{x}_d)}{\partial w_i} = x_{i,d}$

So: $\frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$

Given $(x_d, t_d)_{d \in D}$ find **w** to minimize $\sum_{d \in D} (o_d - t_d)^2$



o_d = observed unit output for x_d

$o_d = \sigma(net_d)$; $net_d = \sum_i w_i x_{i,d}$

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

δ_d error term $t_d - o_d$ multiplied by $o_d(1 - o_d)$ that comes from the derivative of the sigmoid function

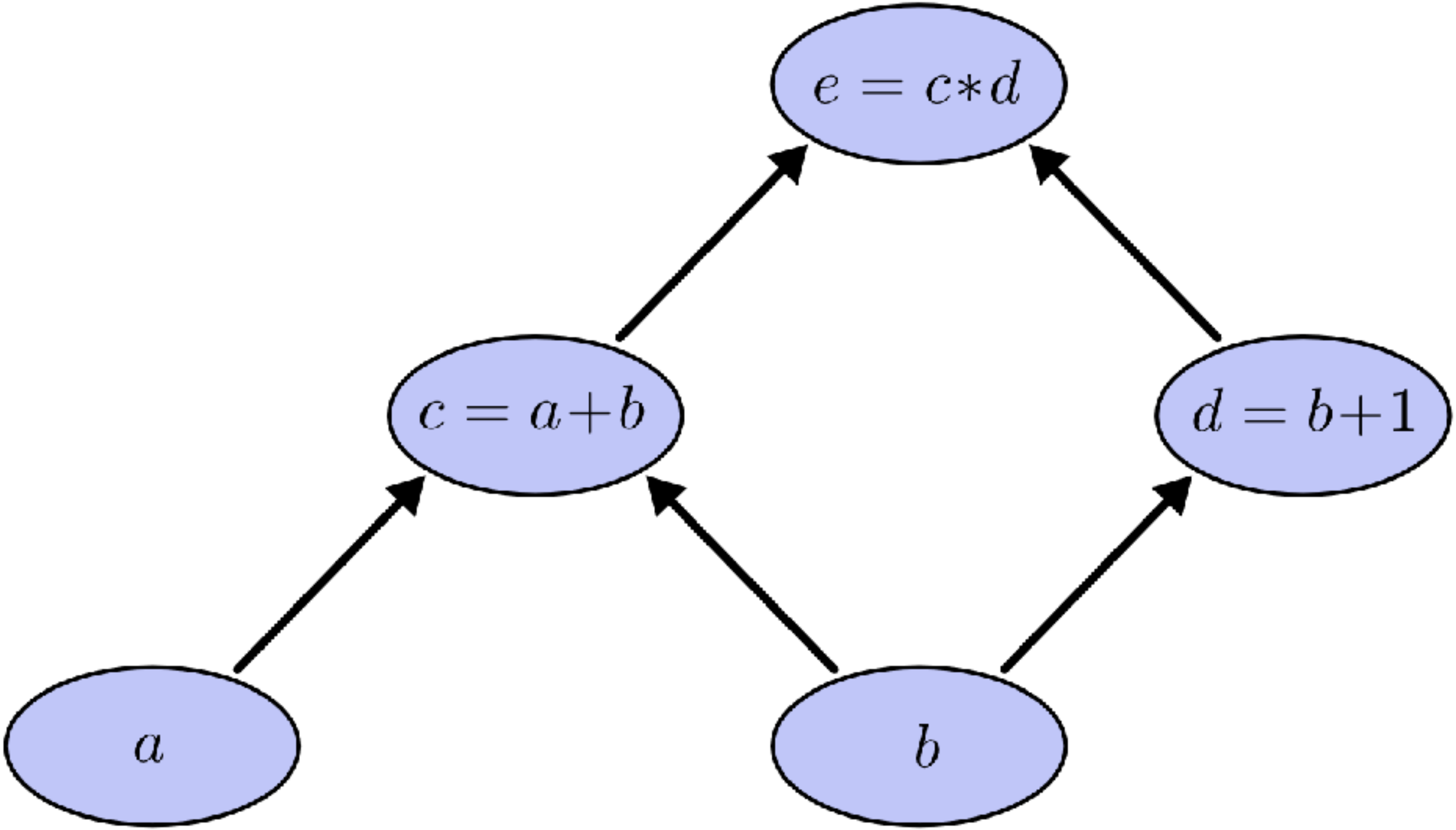
$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} \delta_d \, x_{i,d}$$

Update rule: $w \leftarrow w - \eta \nabla E[w]$

Computation Graphs

Towards Auto-differentiation

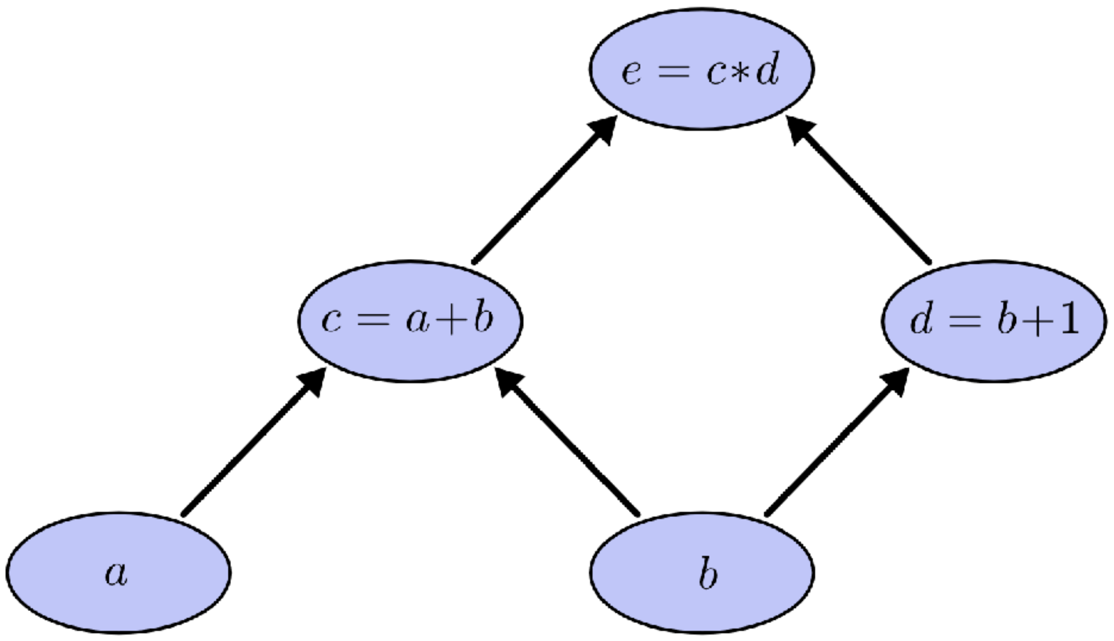
(Autograd)

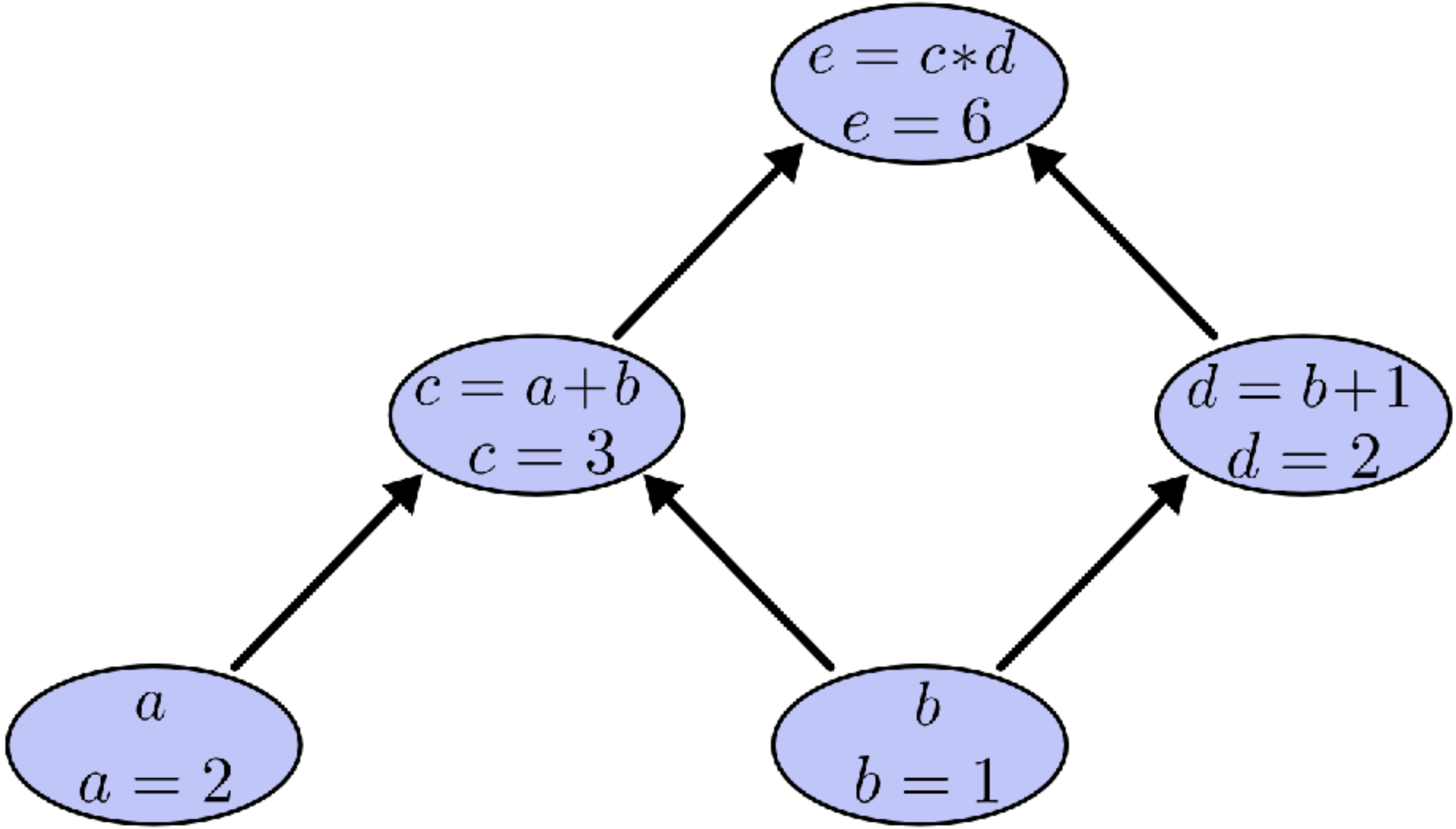


$e = c*d$
$c = a+b$
$d = b+1$
a
b

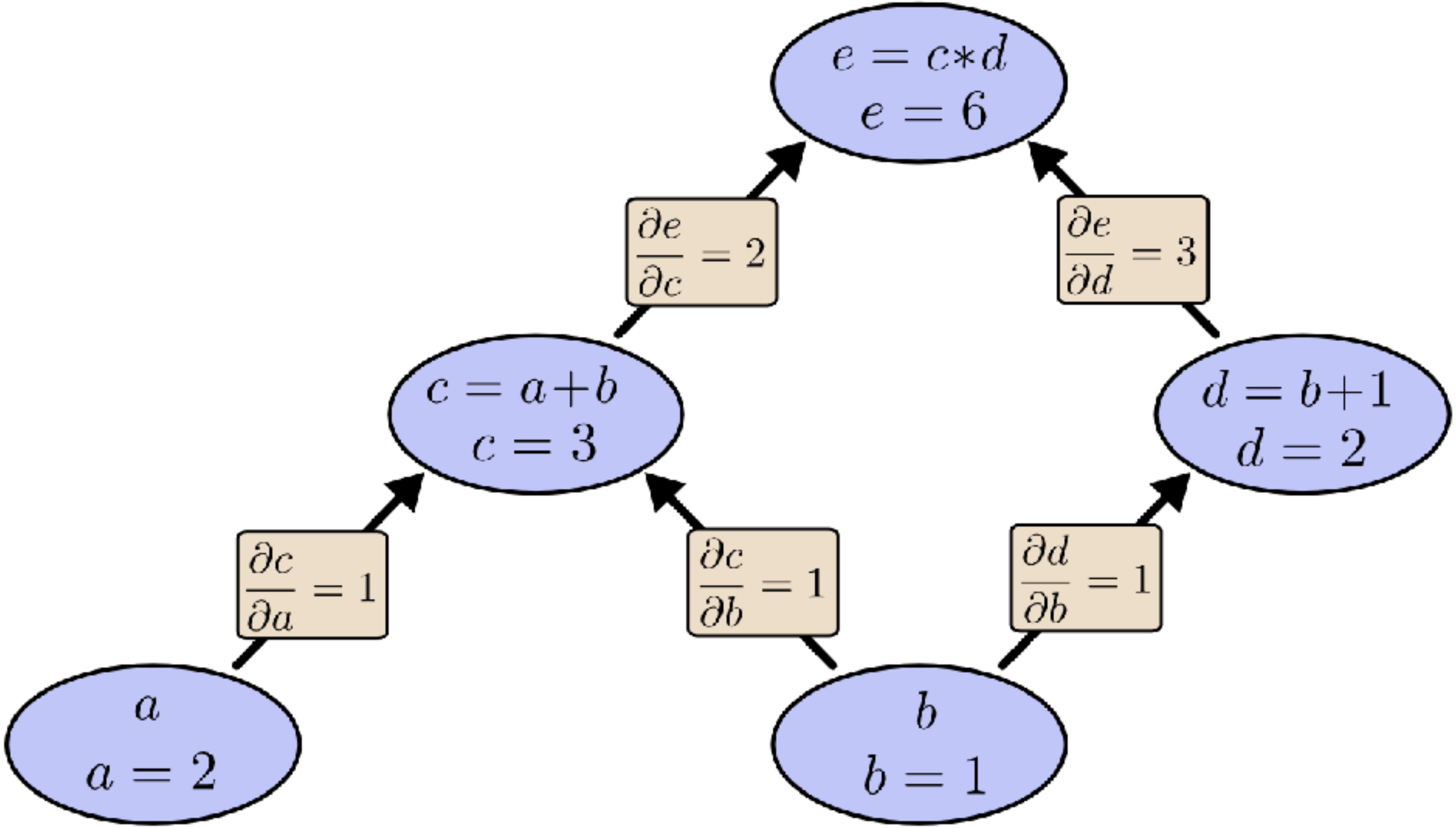
$$\frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$$

$$\frac{\partial}{\partial u}uv = u\frac{\partial v}{\partial u} + v\frac{\partial u}{\partial u} = v$$



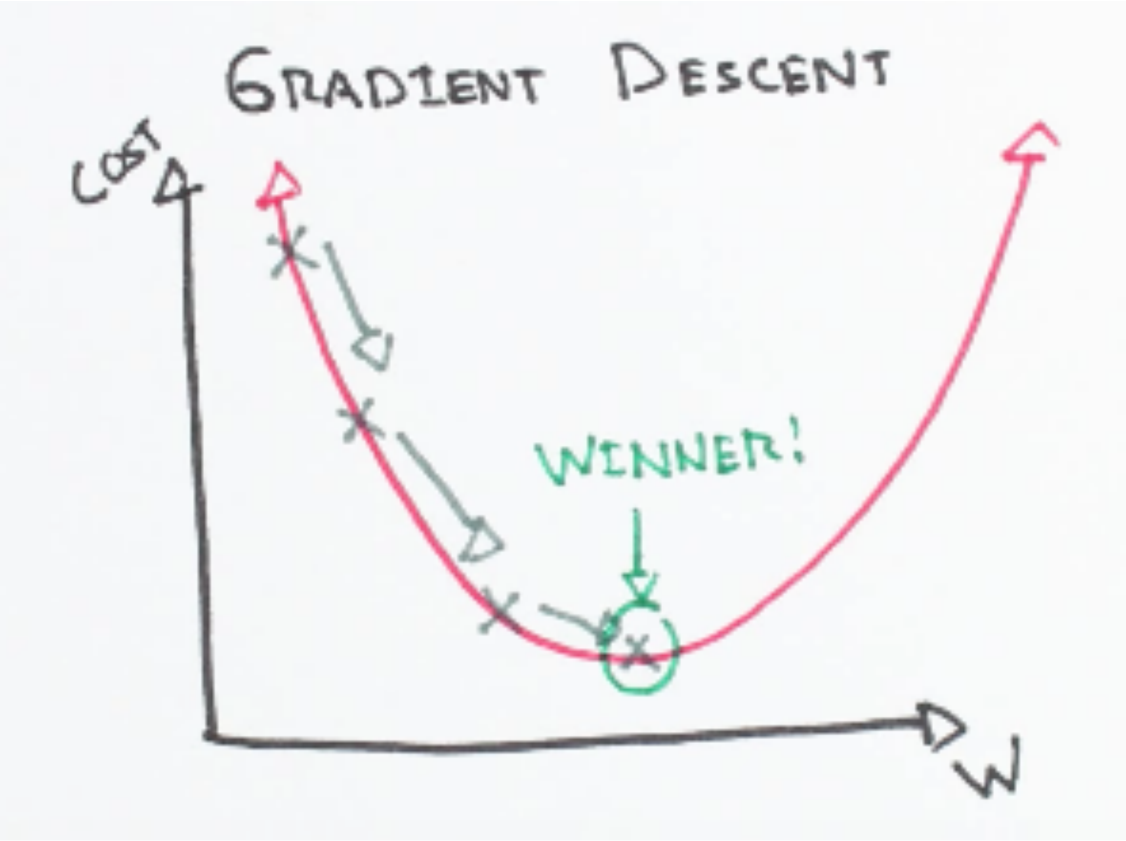


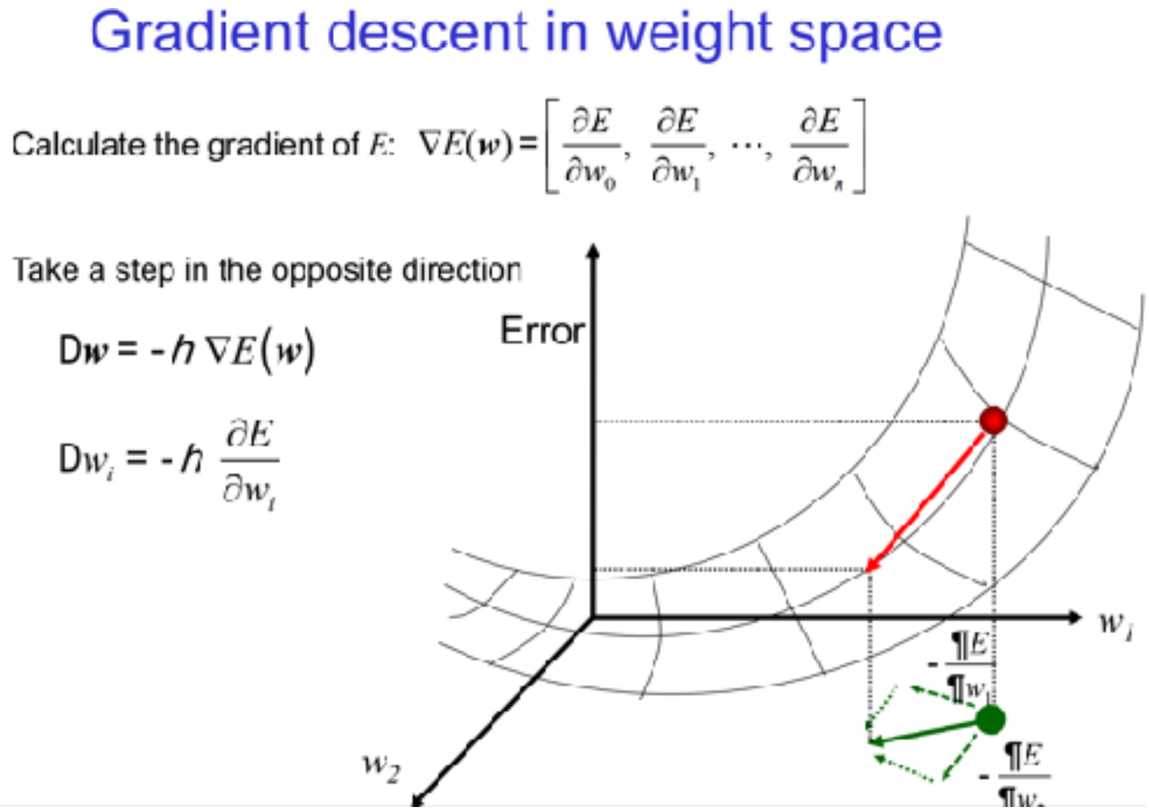
$e = c*d$
$e = 6$
$c = a+b$
$c = 3$
$d = b+1$
$d = 2$
a
$a = 2$
b
$b = 1$

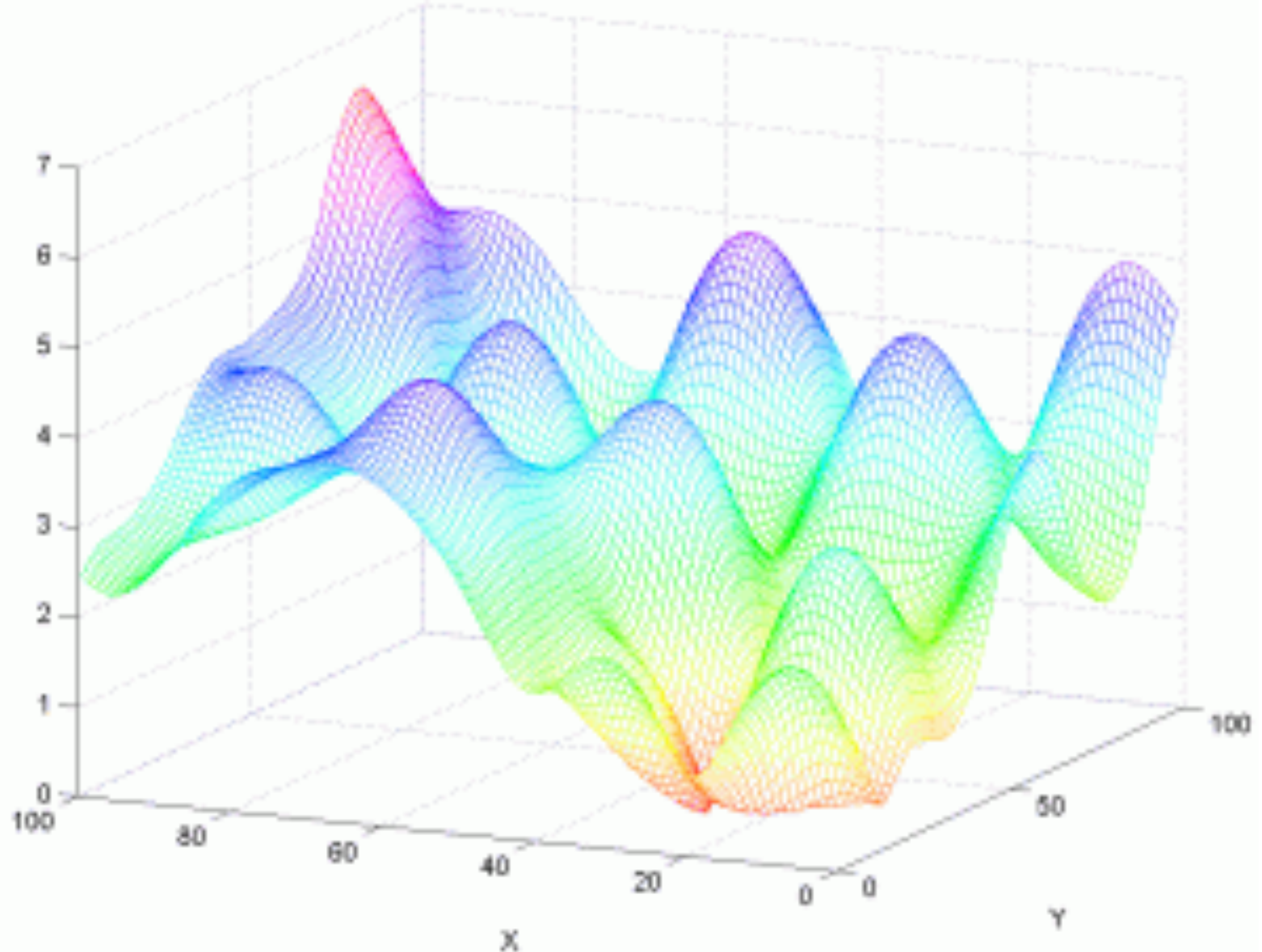


$e = c * d$
$e = 6$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$c = a + b$
$c = 3$
$d = b + 1$
$d = 2$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
a
$a = 2$
b
$b = 1$

Gradient Descent





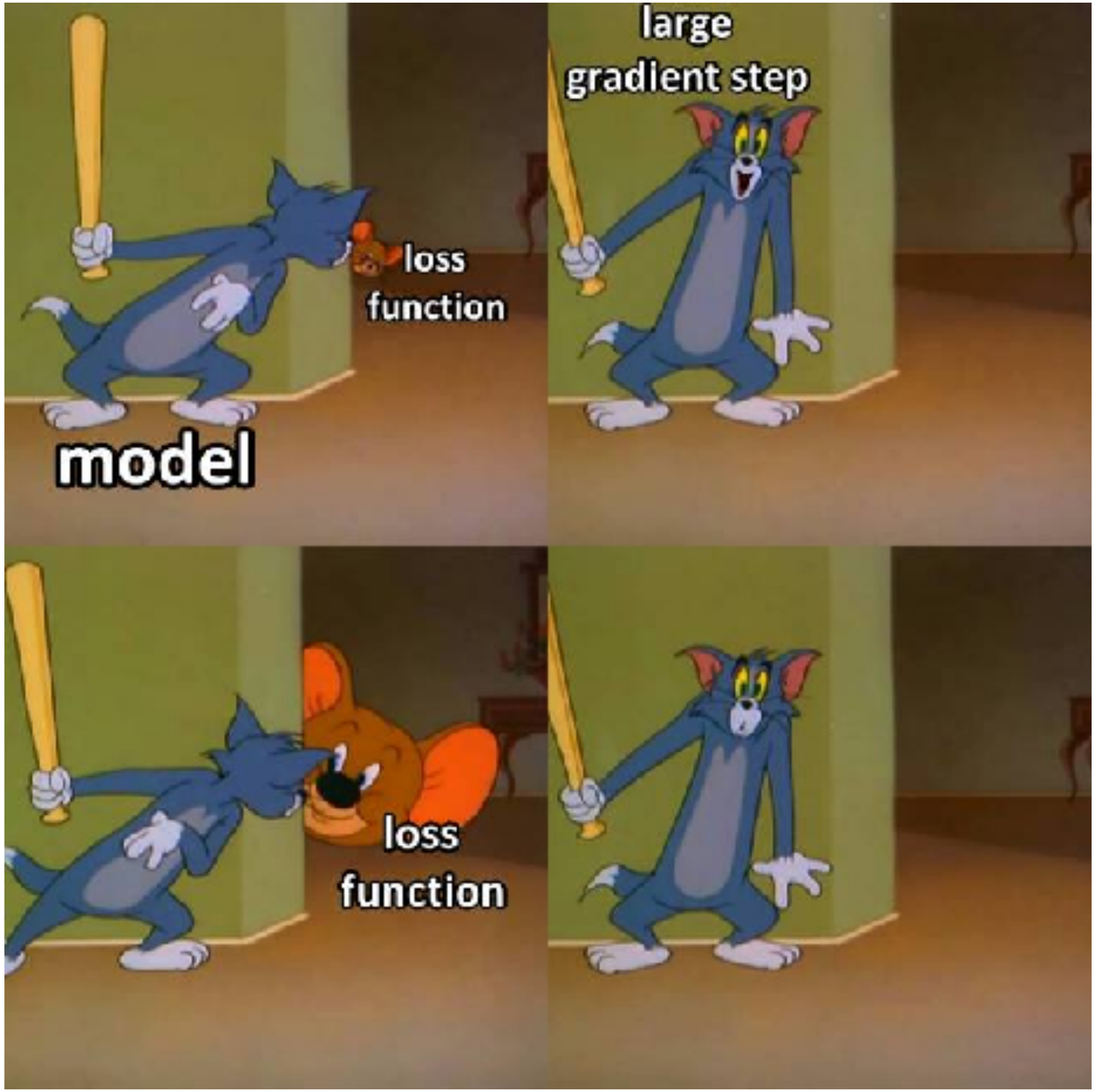


7
6
5
4
3
2
1
0
100
80
60
40
20
0
X
0
50
100
Y

Does it work?


Training IoU
Validation IoU
IoU
Epoch
Looks good, let's train for a few more epochs!
Training IoU
Validation IoU
IoU
Epoch

Gradient!!!



Why Doesn't Gradient Tell You Everything

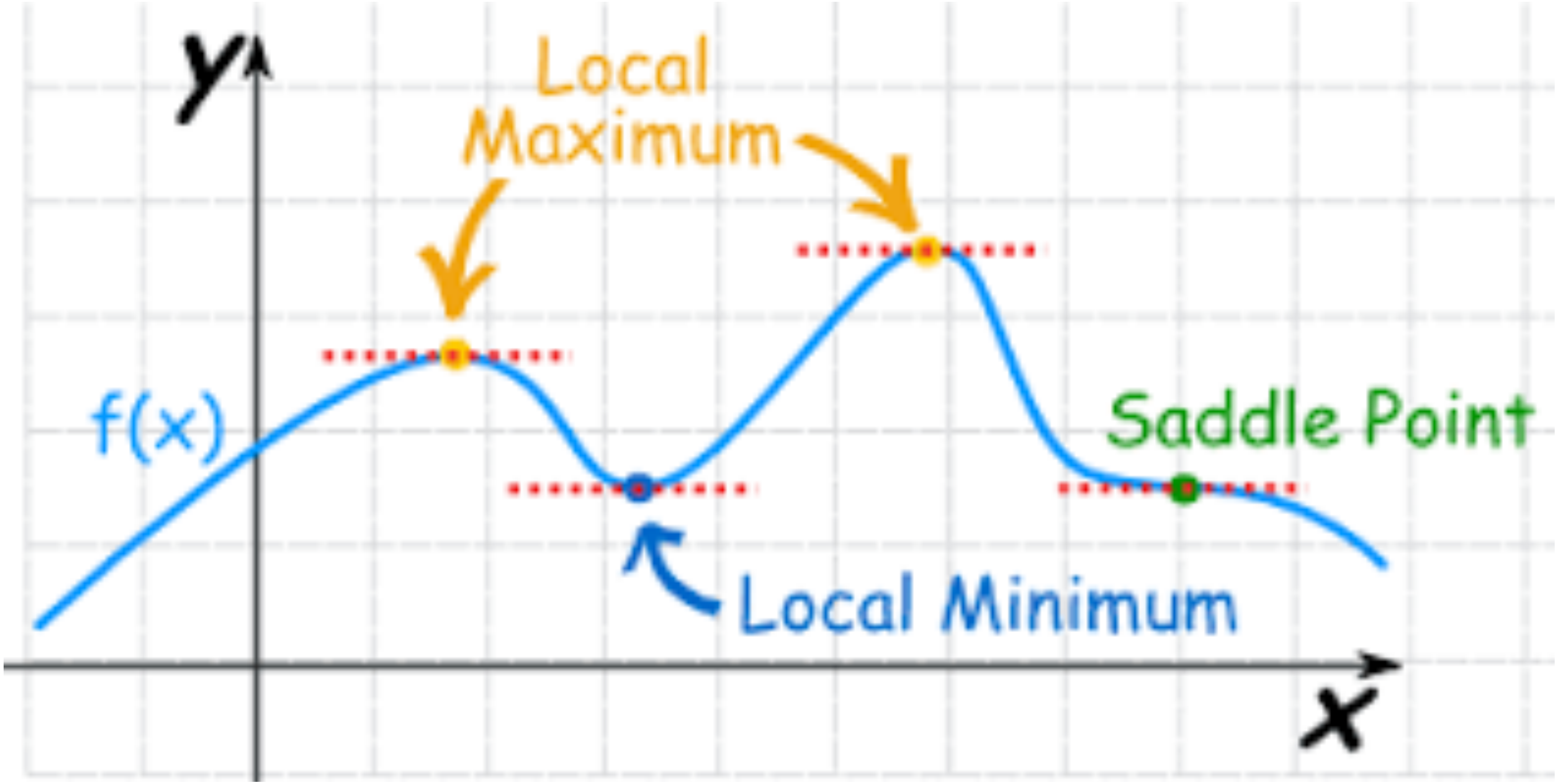
Taylor Series for $f(x)$ about $x = a$ is,

Taylor Series

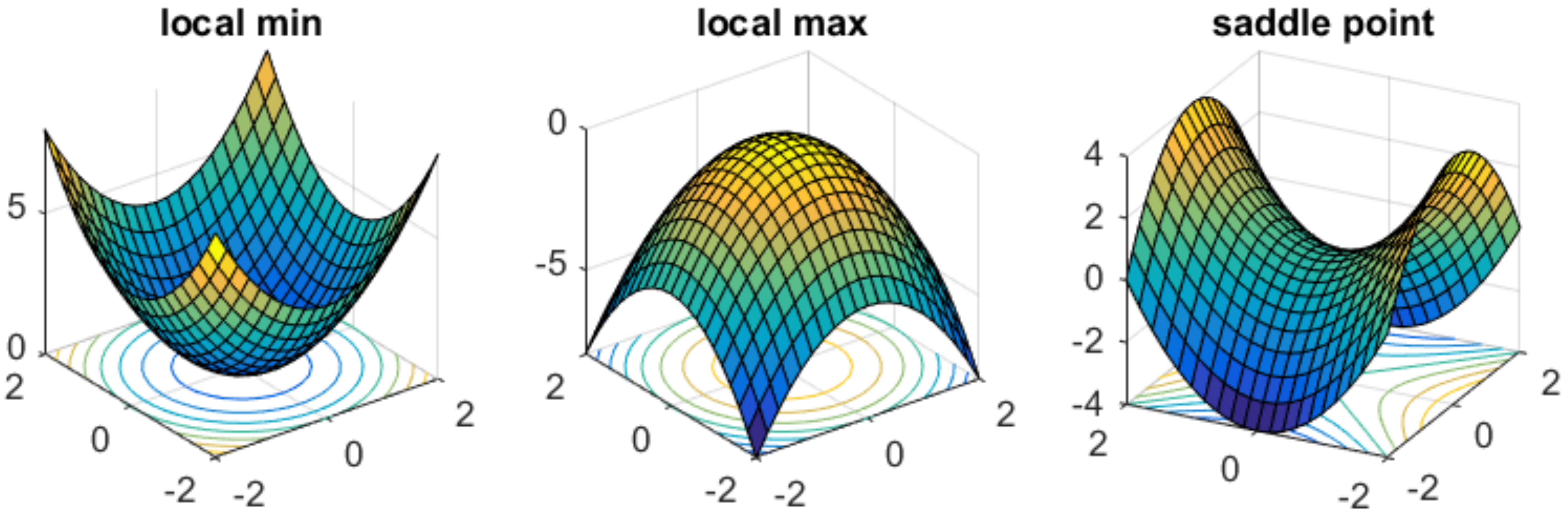
$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$$= f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

Local Min & Saddle Points!



Saddle Point



Hessian Matrix

$$\mathbf{H} = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\[2.2ex] \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\[2.2ex] \vdots & \vdots & \ddots & \vdots \\[2.2ex] \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

Momentum

Image 2: SGD without momentum

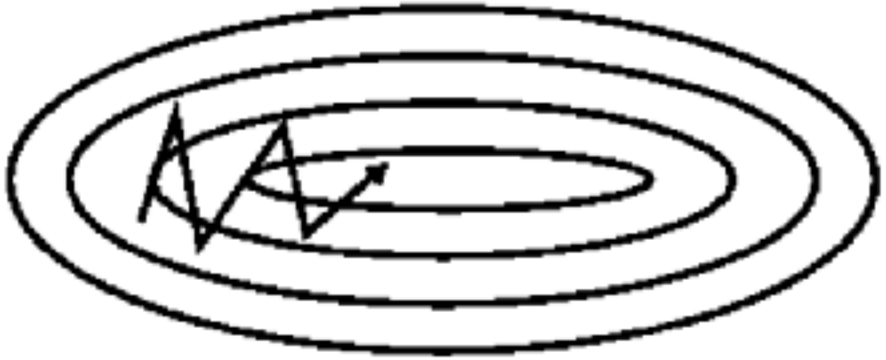


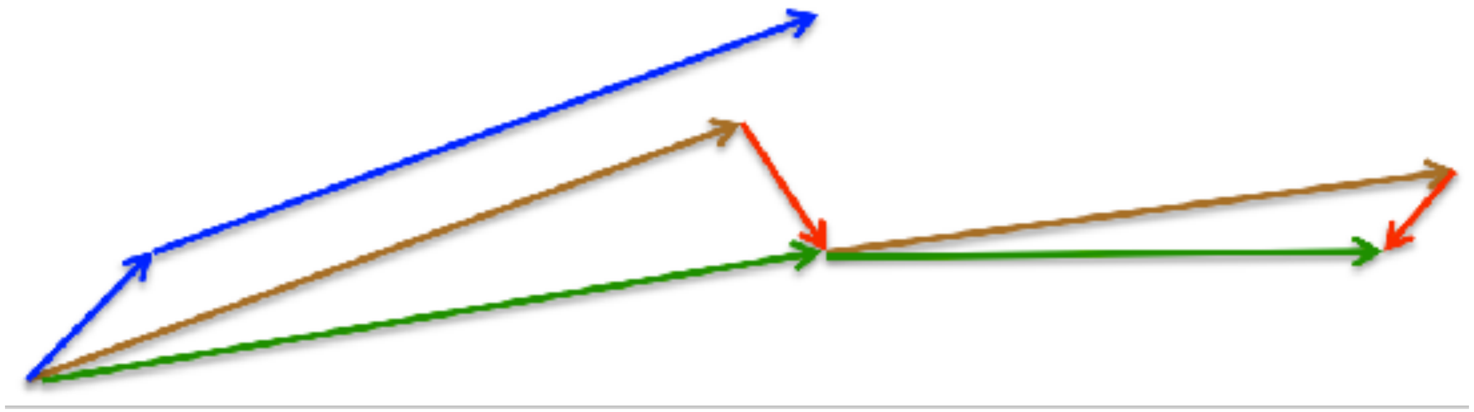
Image 3: SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

However, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory.

Can we have a smarter ball that slows down before going up again?

Nesterov Momentum



$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

Performs “look-ahead” by estimating the updated parameters with the current momentum only.
Estimate the gradient based on the momentum updated parameters.

Adagrad

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}).$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

$G_t \in \mathbb{R}^{d \times d}$ here is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t [12], while ϵ is a smoothing term that avoids division by zero (usually on the order of $1e-8$).

Problem: Sum of squares.

RMSprop

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}).$$

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2.$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t.$$

Instead of accumulating all past squared gradients, RMSprop restricts the window of accumulated past gradients to some fixed size.

Adam

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

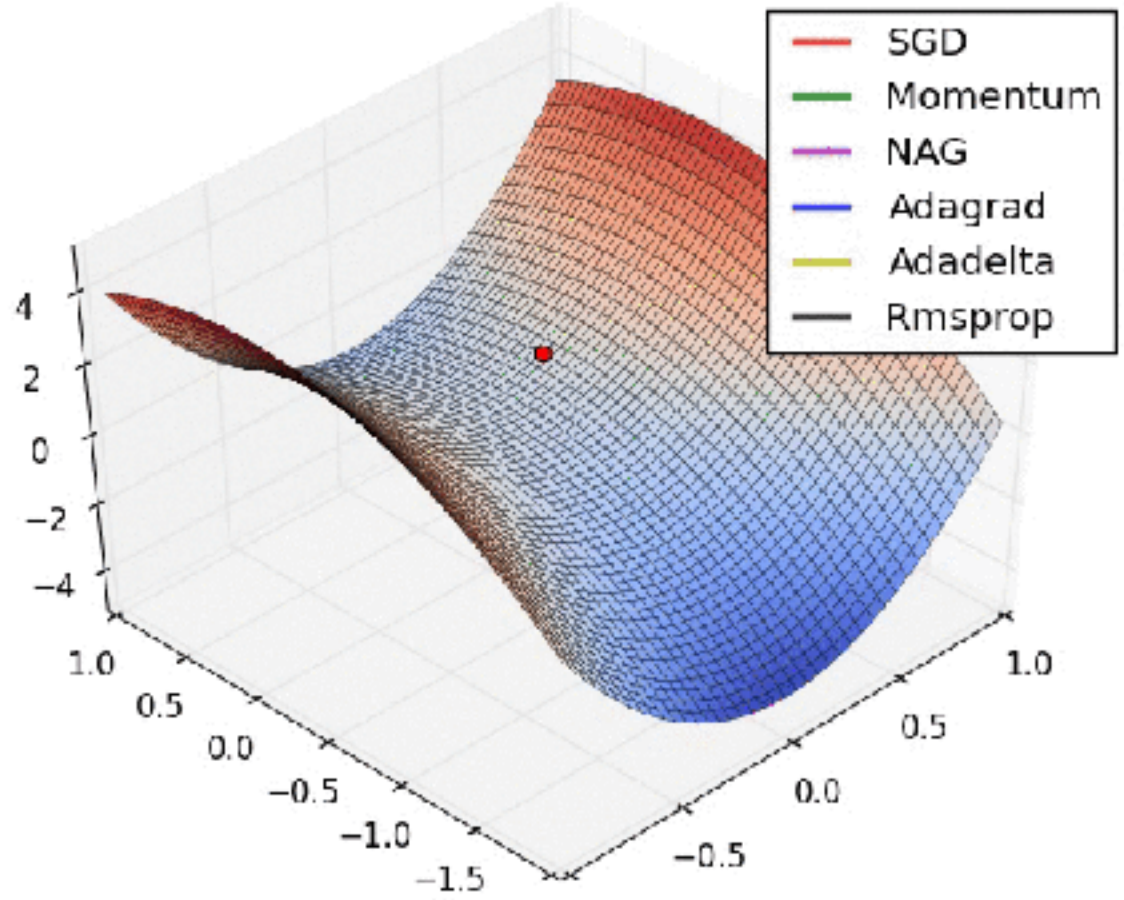
m_t, v_t initialized with 0

This step corrects the bias toward 0

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

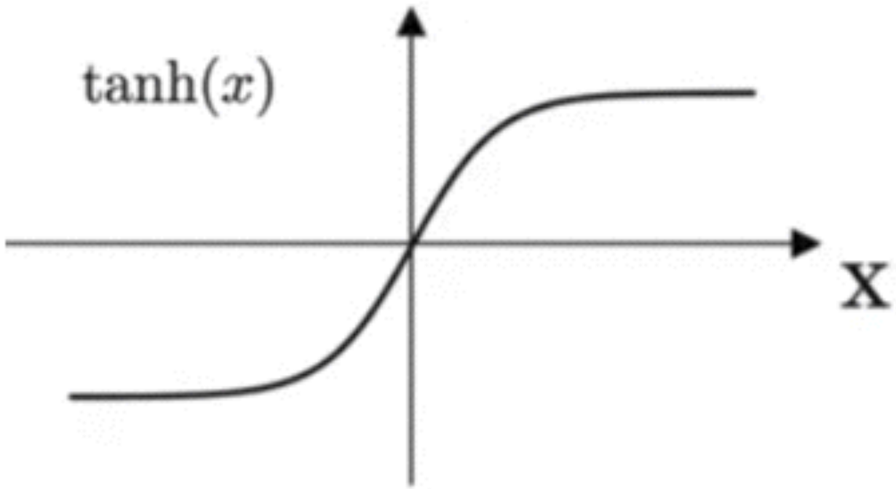
Adam is basically RMSprop with momentum.

Hessian Free Optimization Methods

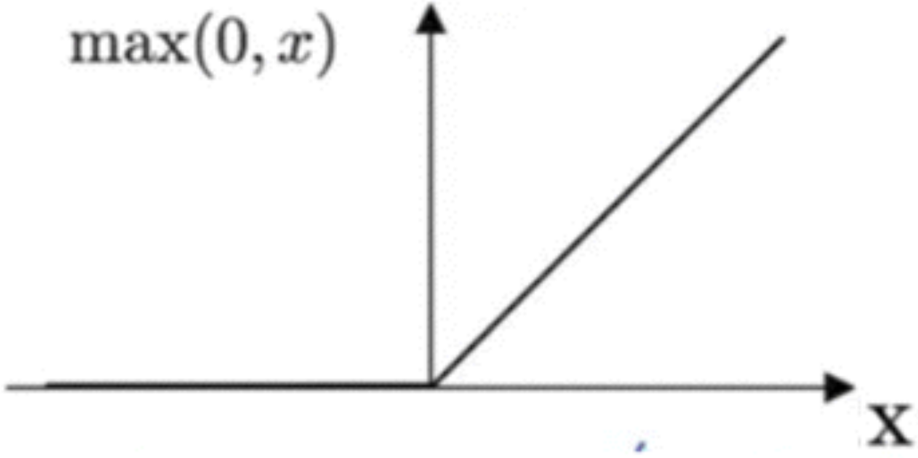


Vanishing & Exploding GRADIENT!

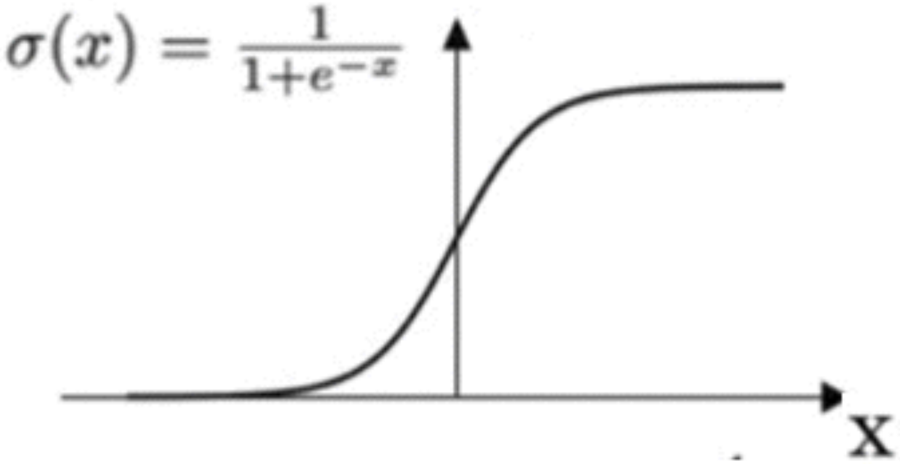
Hyper Tangent Function



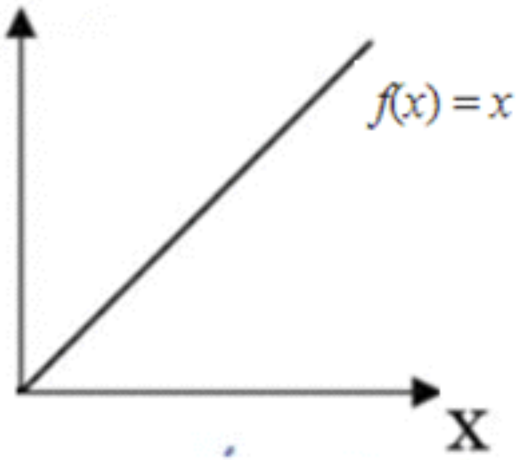
ReLU Function



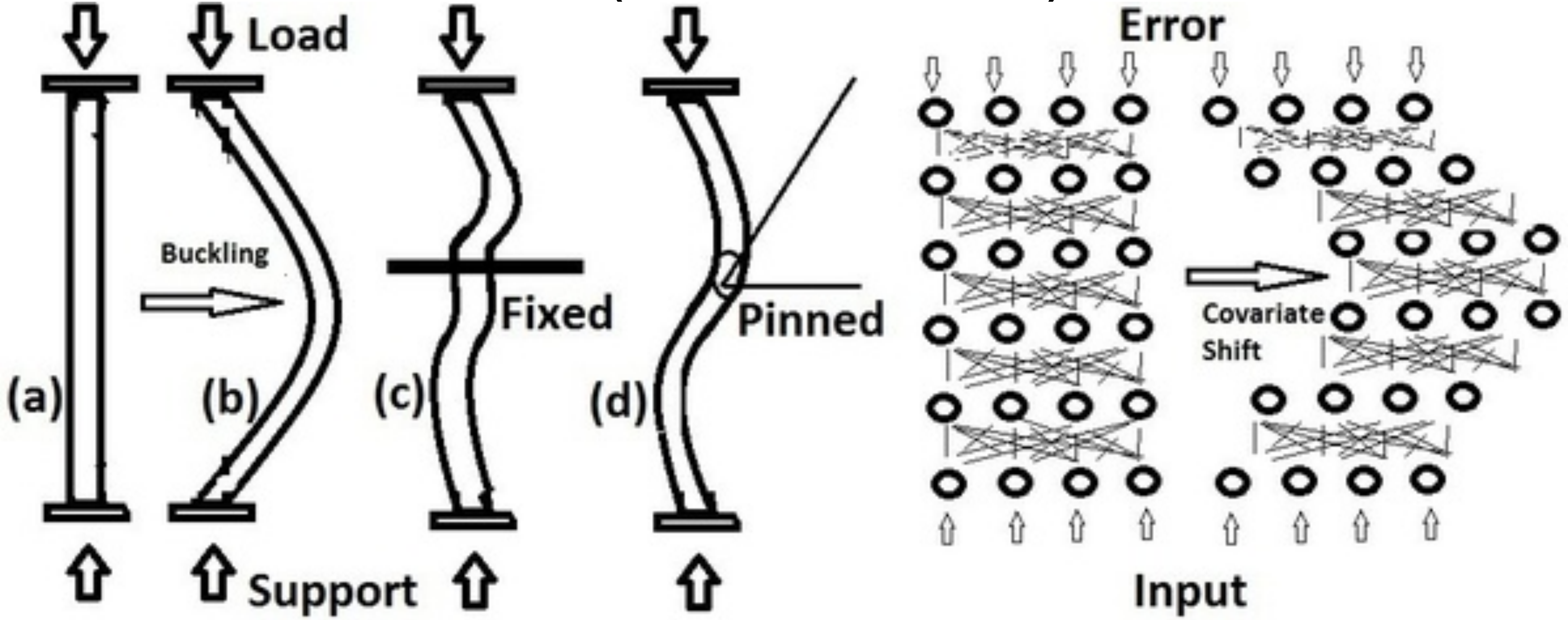
Sigmoid Function



Identity Function



Covariant Shift (BatchNorm)

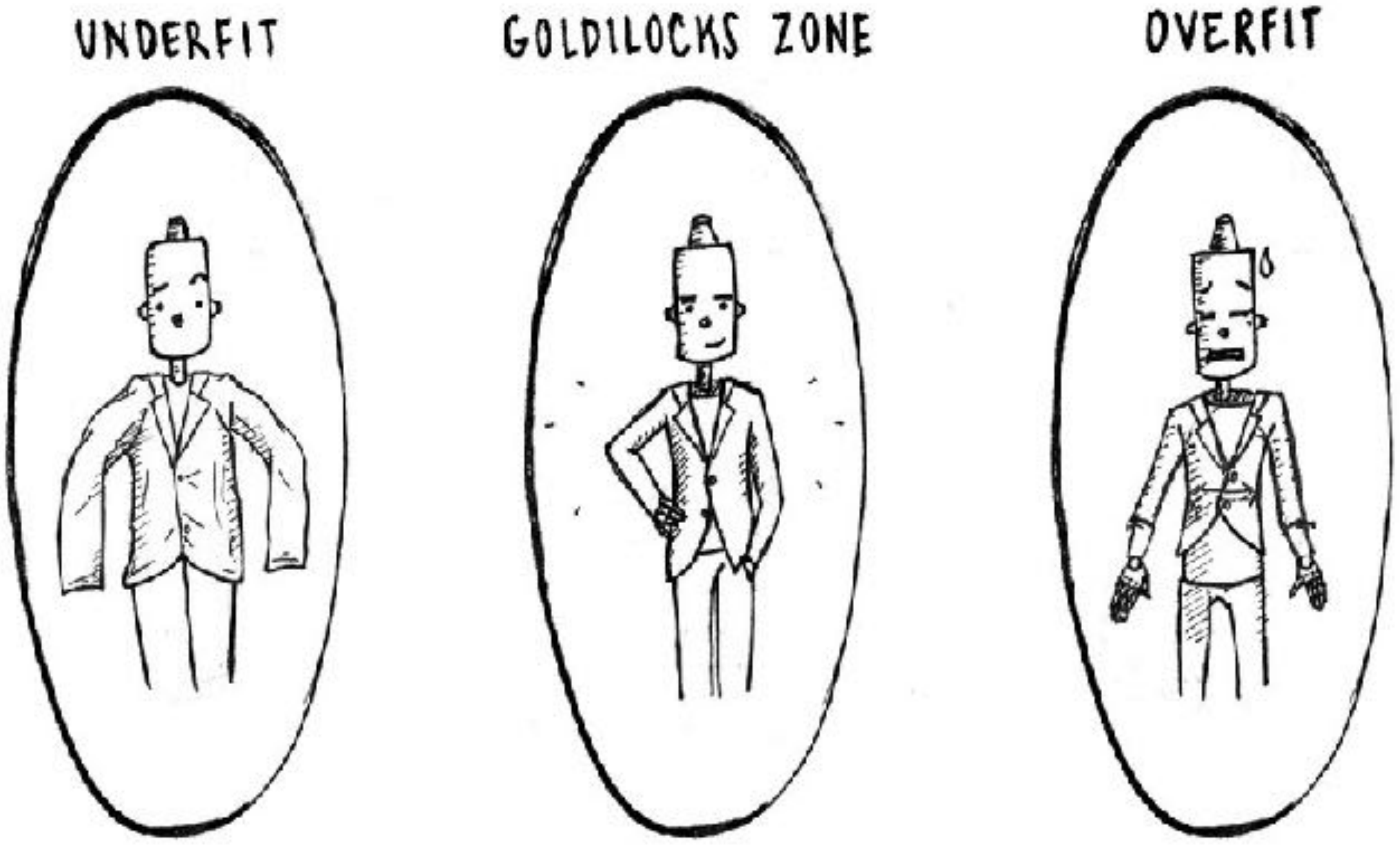


For both, Buckling or Co-Variate Shift a small perturbation leads to a large change in the later.

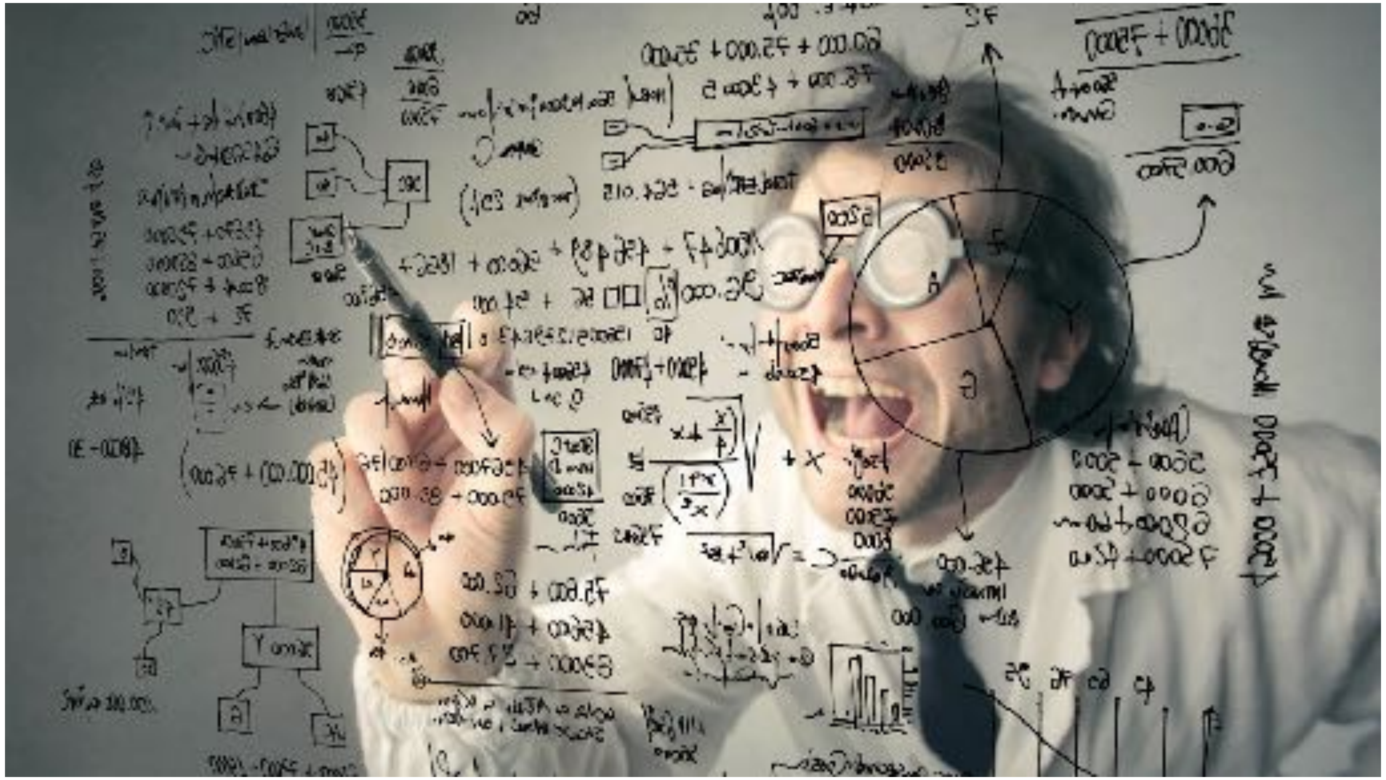
Debiprasad Ghosh, PhD, Uses AI in Mechanics

MACHINE LEARNING GENERALIZATION

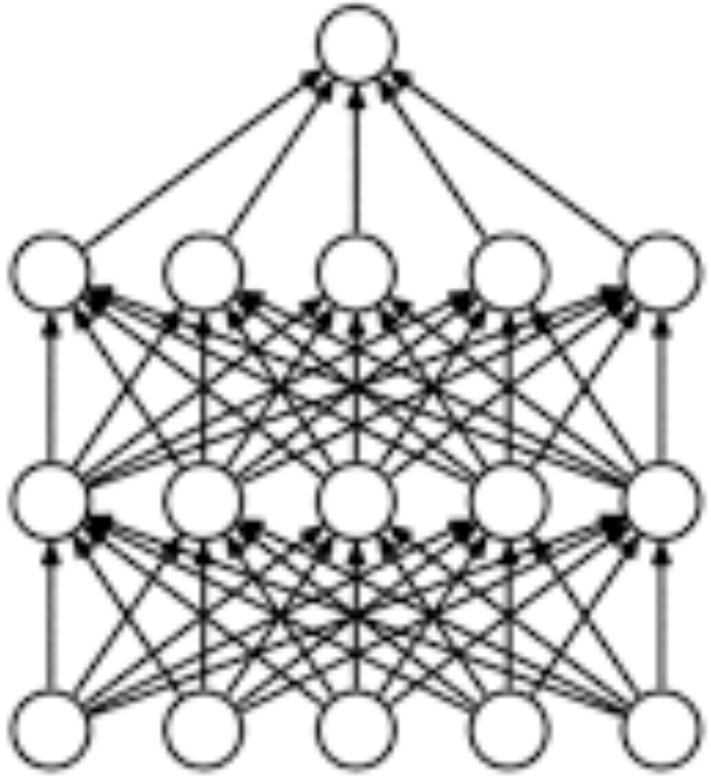
FINDING THE PERFECT FIT



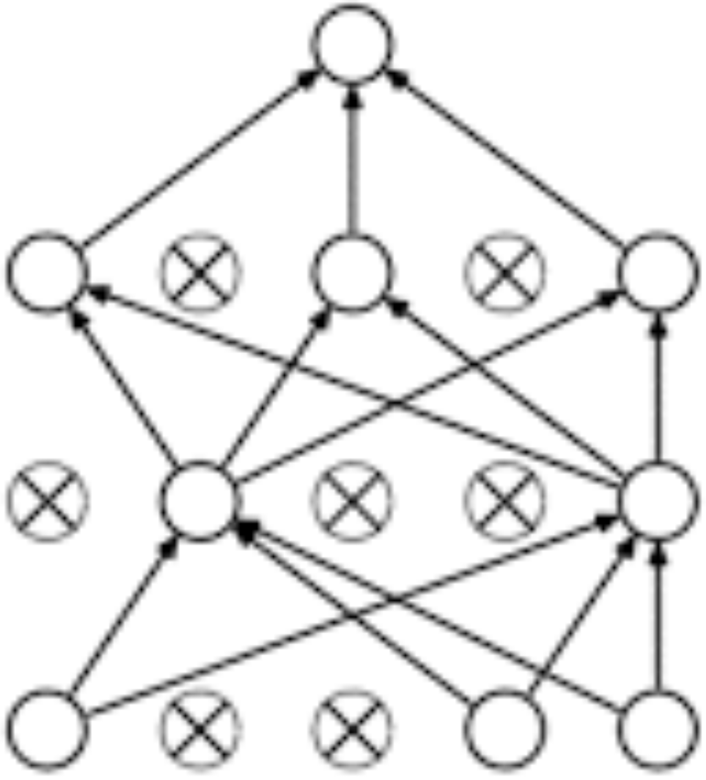
GET MORE DATA!!!!



Dropout



(a) Standard Neural Net

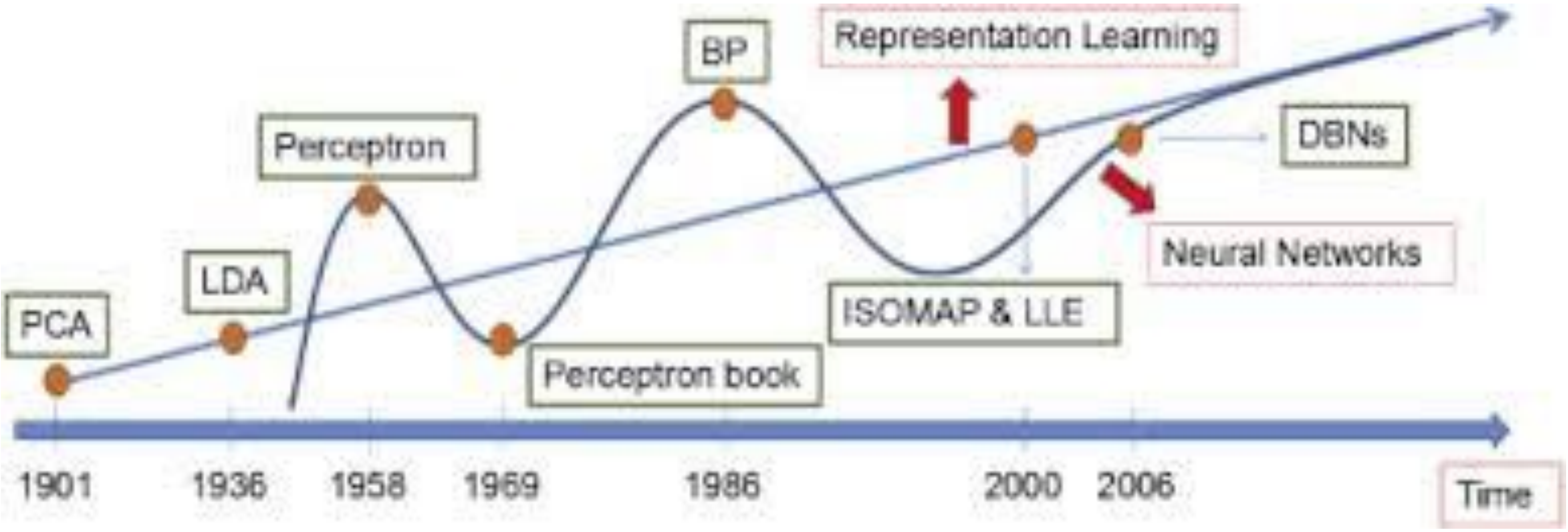


(b) After applying dropout.

A Shallow Intro to Deep Learning

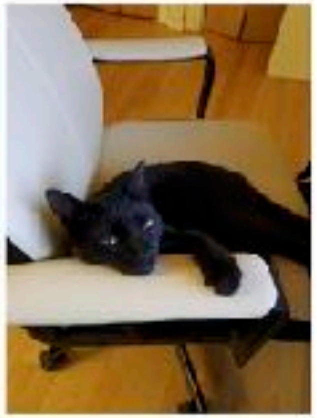
Yue Wu

Why CNN?



Representation Learning
BP
Perceptron
DBNs
Neural Networks
LDA
ISOMAP & LLE
PCA
Perceptron book
1901
1936
1958
1969
1986
2000
2006
Time

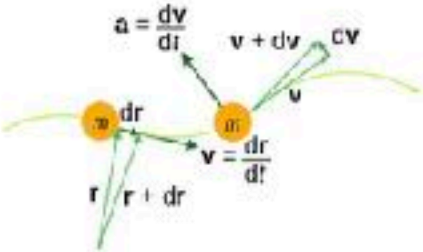
Things...

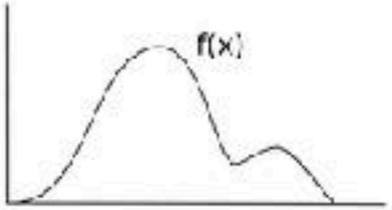


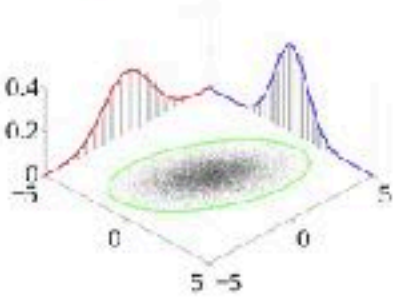
My heart beats as if the world is dripping,
you may not feel the love but i do its a heart
breaking moment of your life. enjoy the times
that we have, it might not sound good but
one thing it rhymes it might not be romantic
but i think it is great,the best rhyme i've ever
heard.

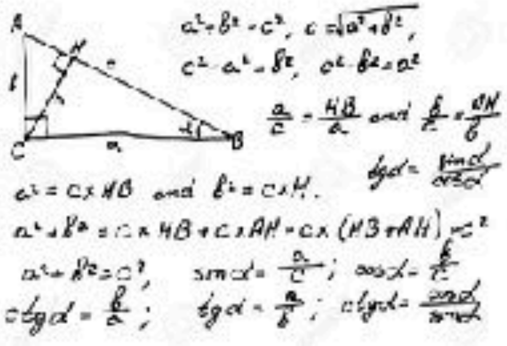
Representation

Engineering Knowledge...





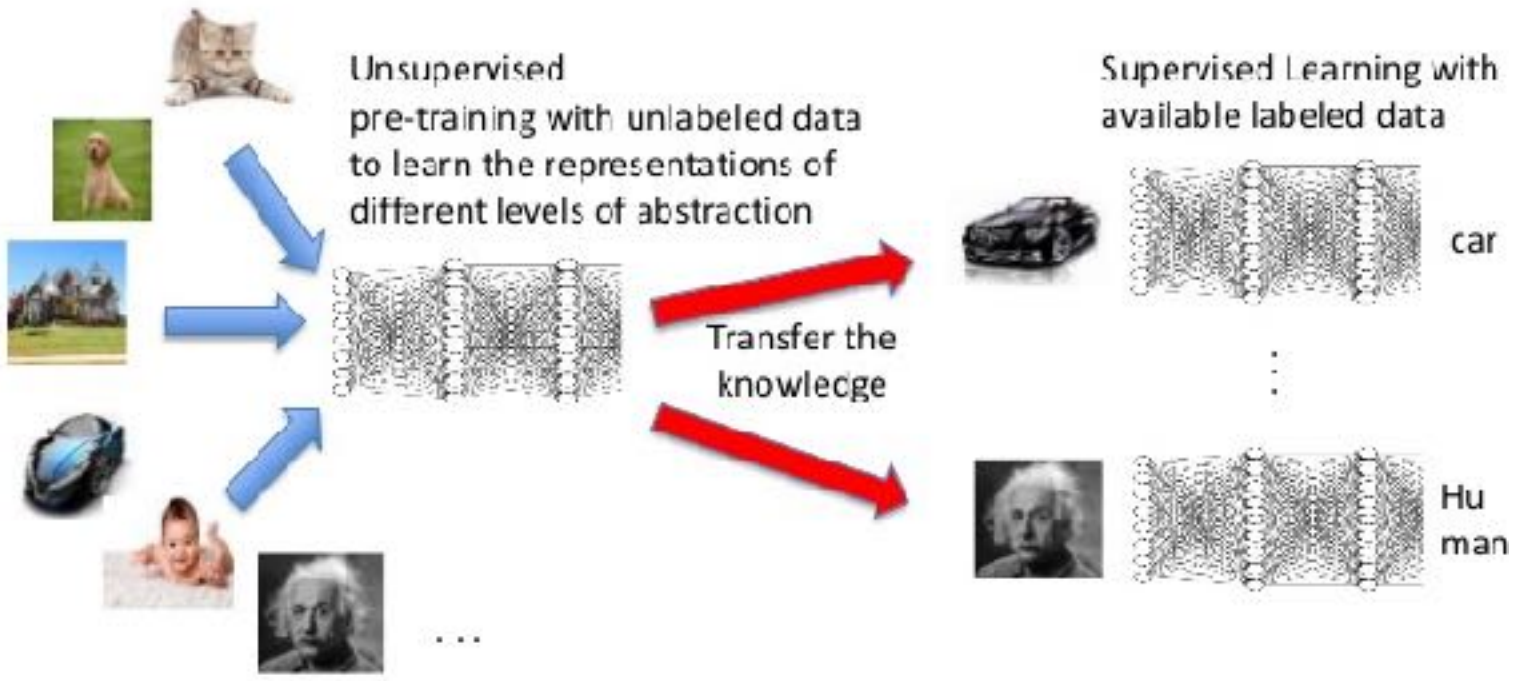




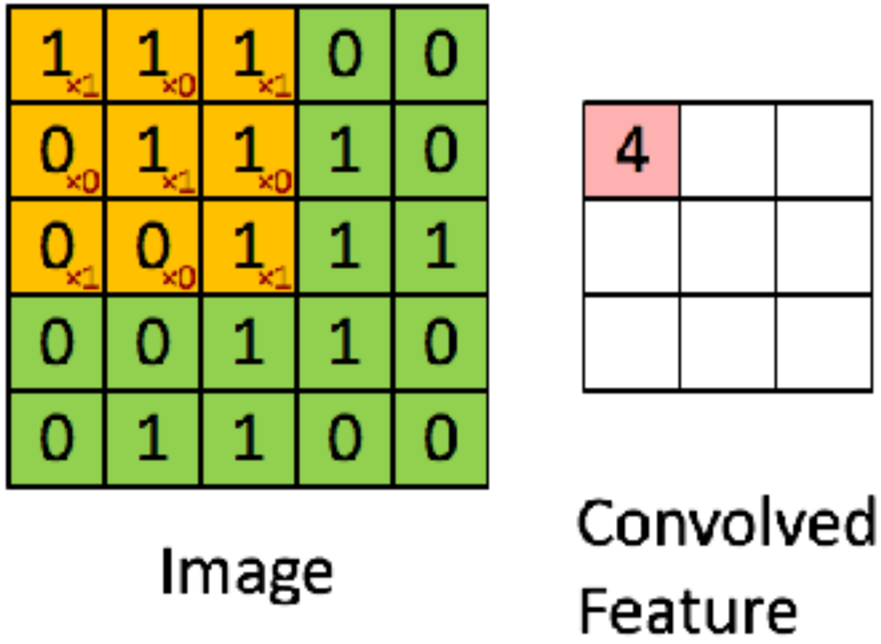
Motivations for Feature/Representation learning

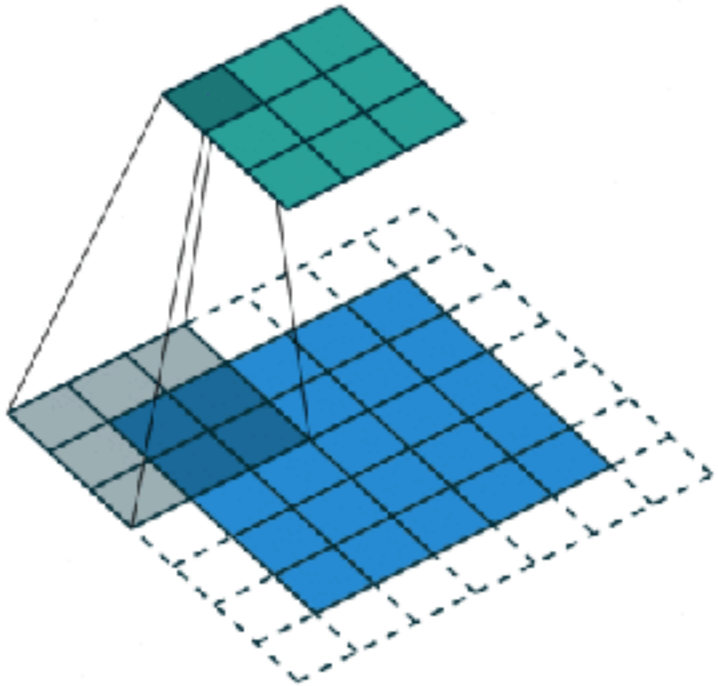
5. Transfer Learning

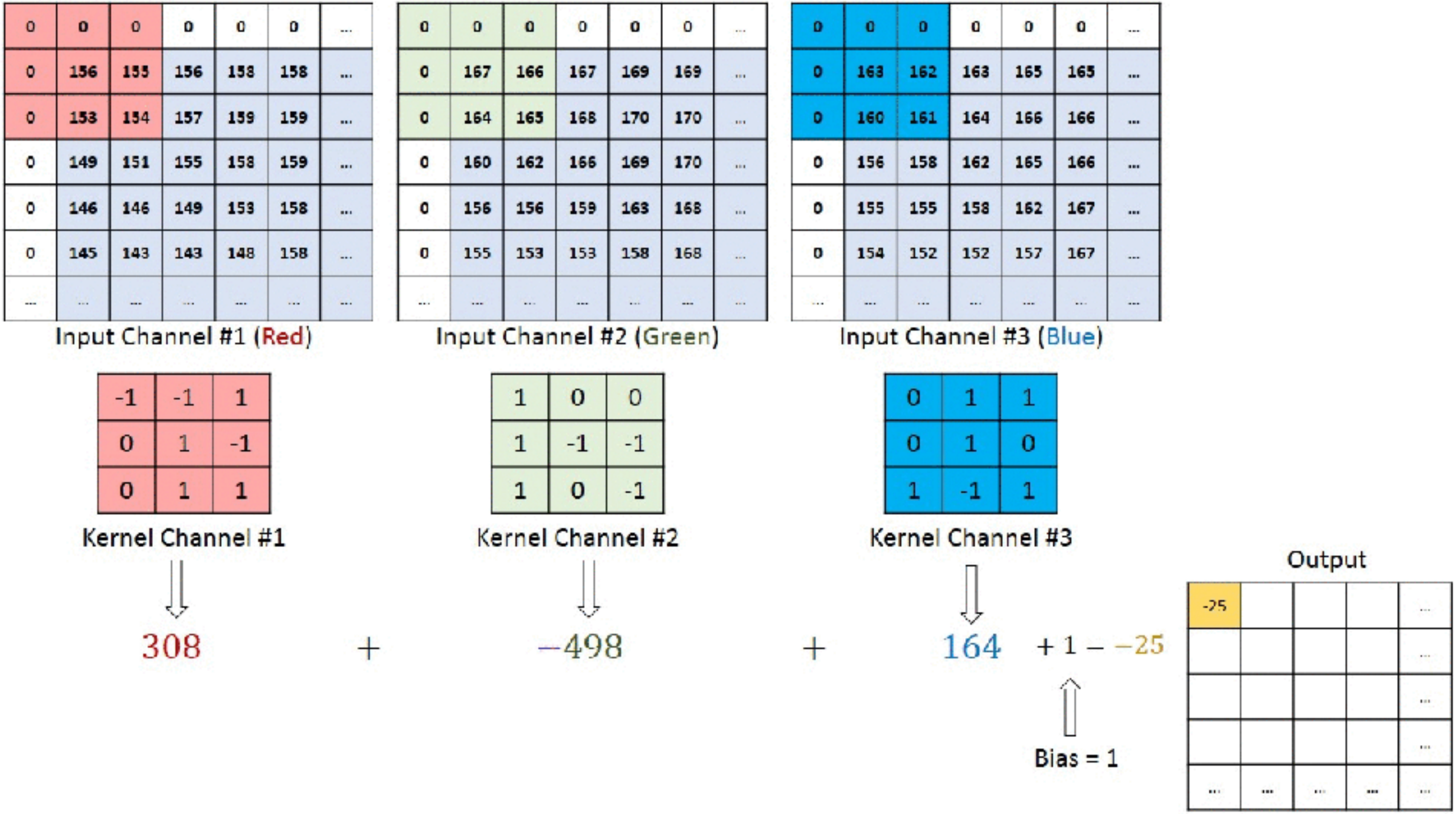
Example: Image recognition model



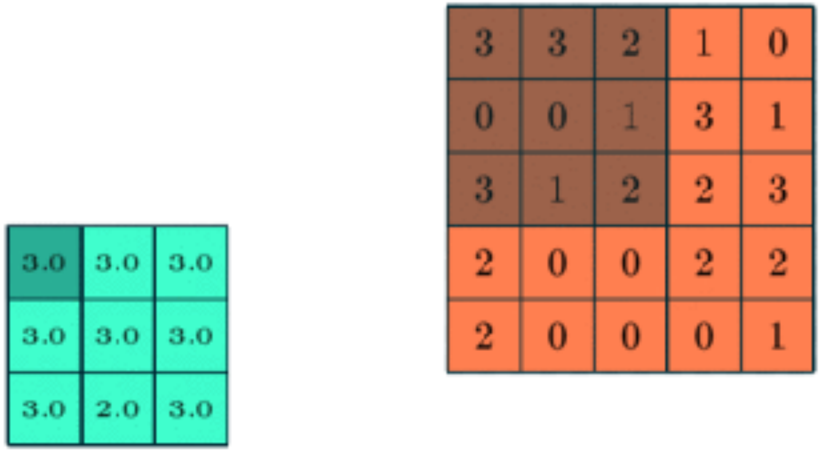
Convolution!

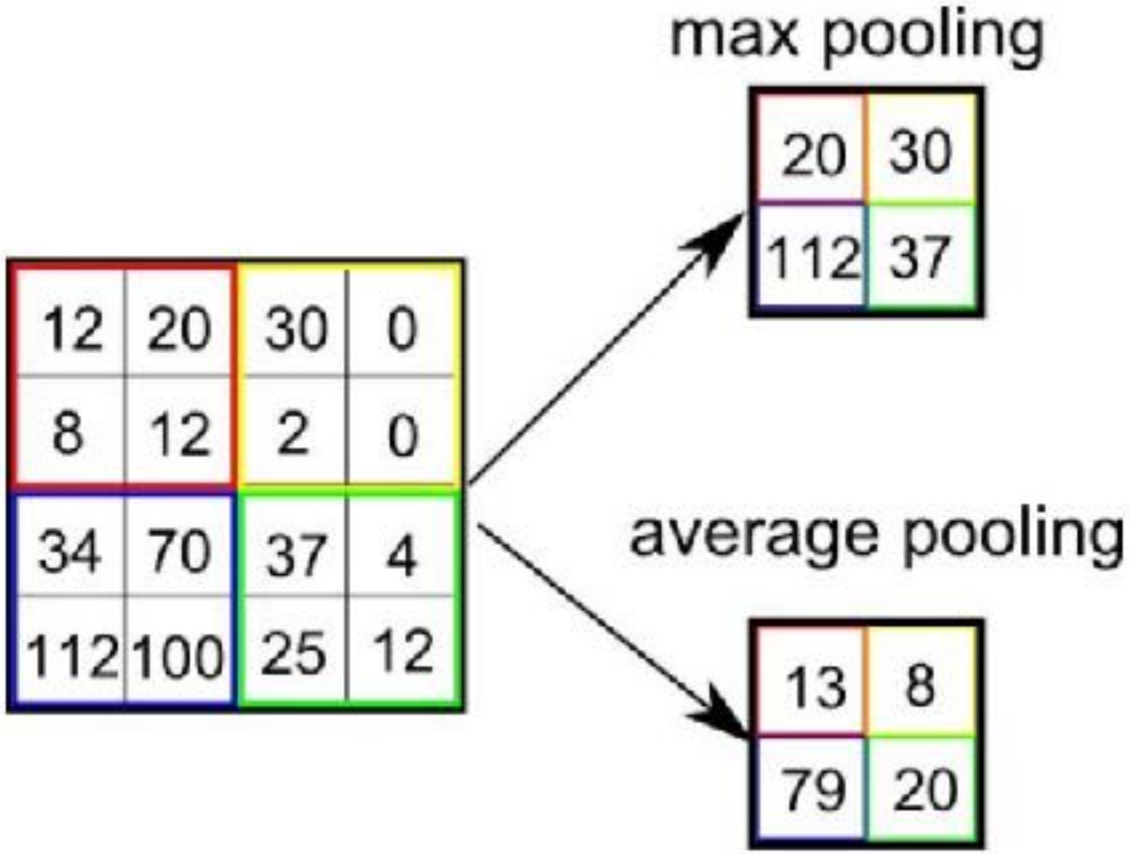




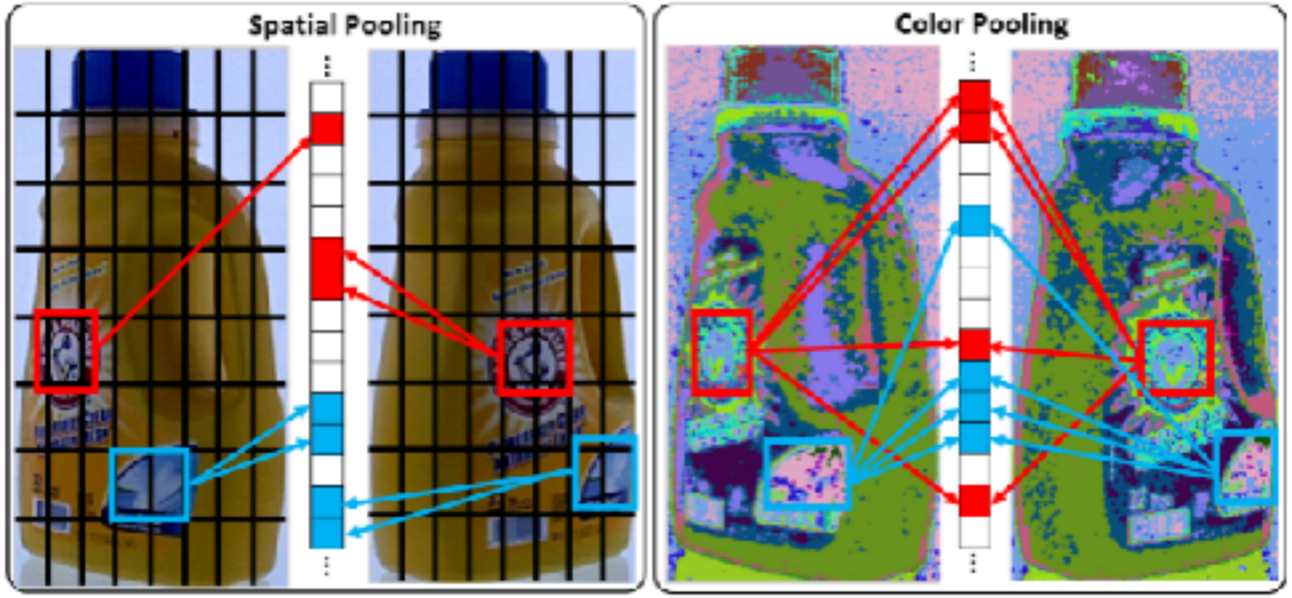

Input Channel #1 (Red)
Input Channel #2 (Green)
Input Channel #3 (Blue)
Kernel Channel #1
Kernel Channel #2
Kernel Channel #3
308
+
−498
+
164
+ 1 = −25
Bias = 1
Output
-25

Pooling!

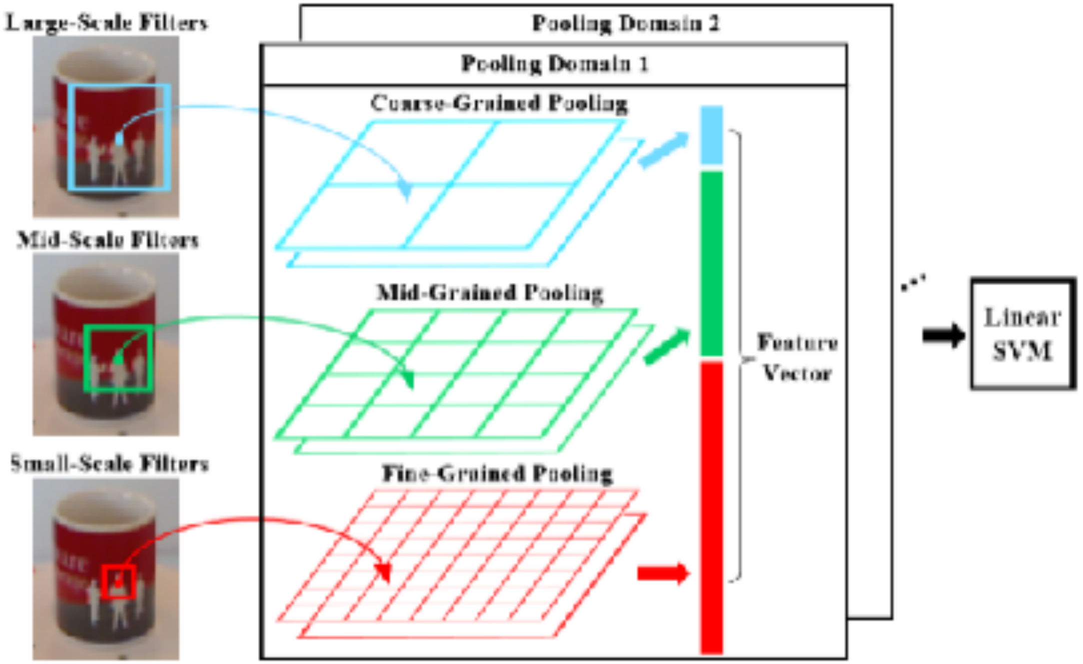

3.0 3.0 3.0
3.0 3.0 3.0
3.0 2.0 3.0
3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1


12 20 30 0
8 12 2 0
34 70 37 4
112 100 25 12
max pooling
20 30
112 37
average pooling
13 8
79 20

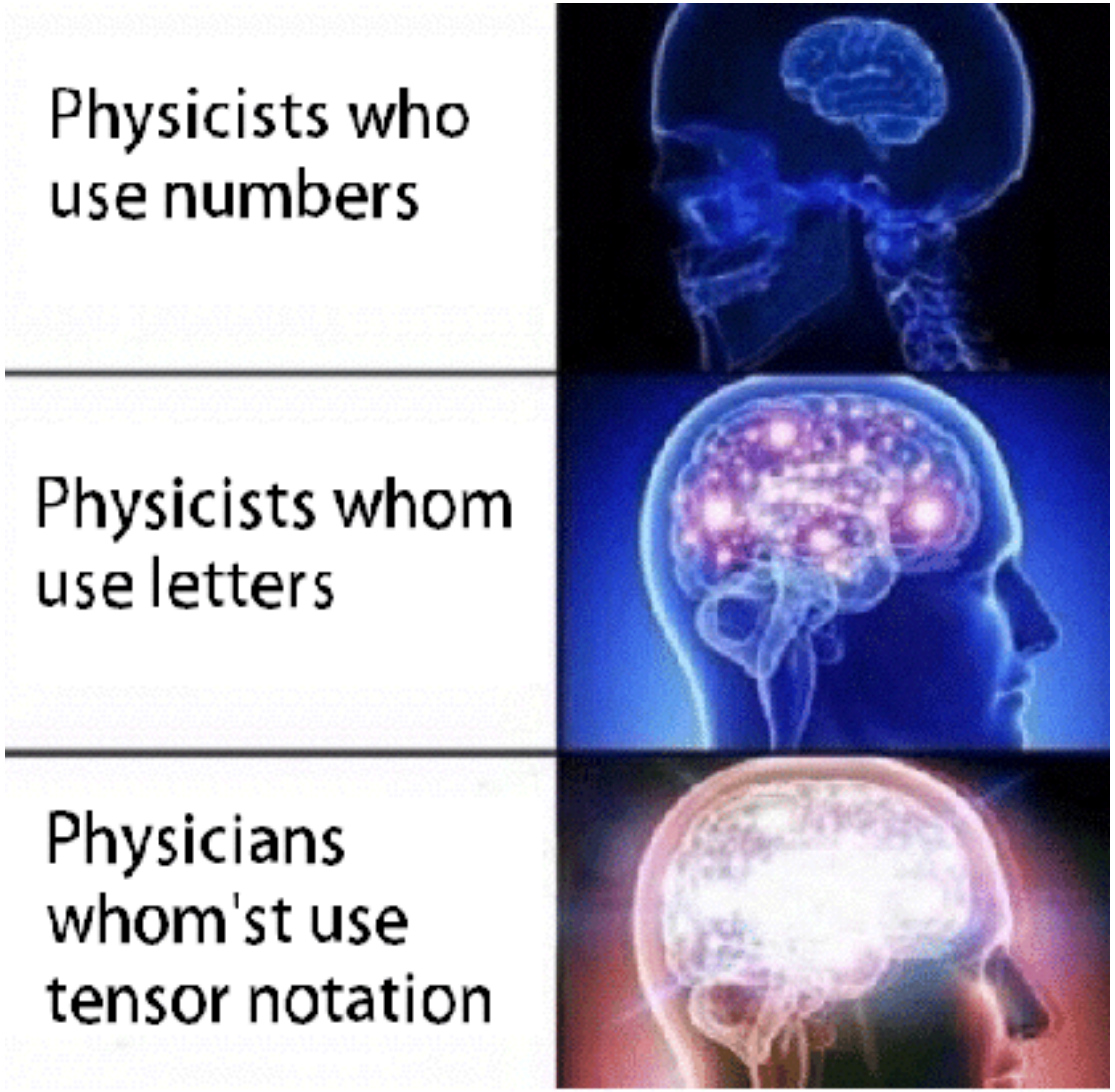
Pooling

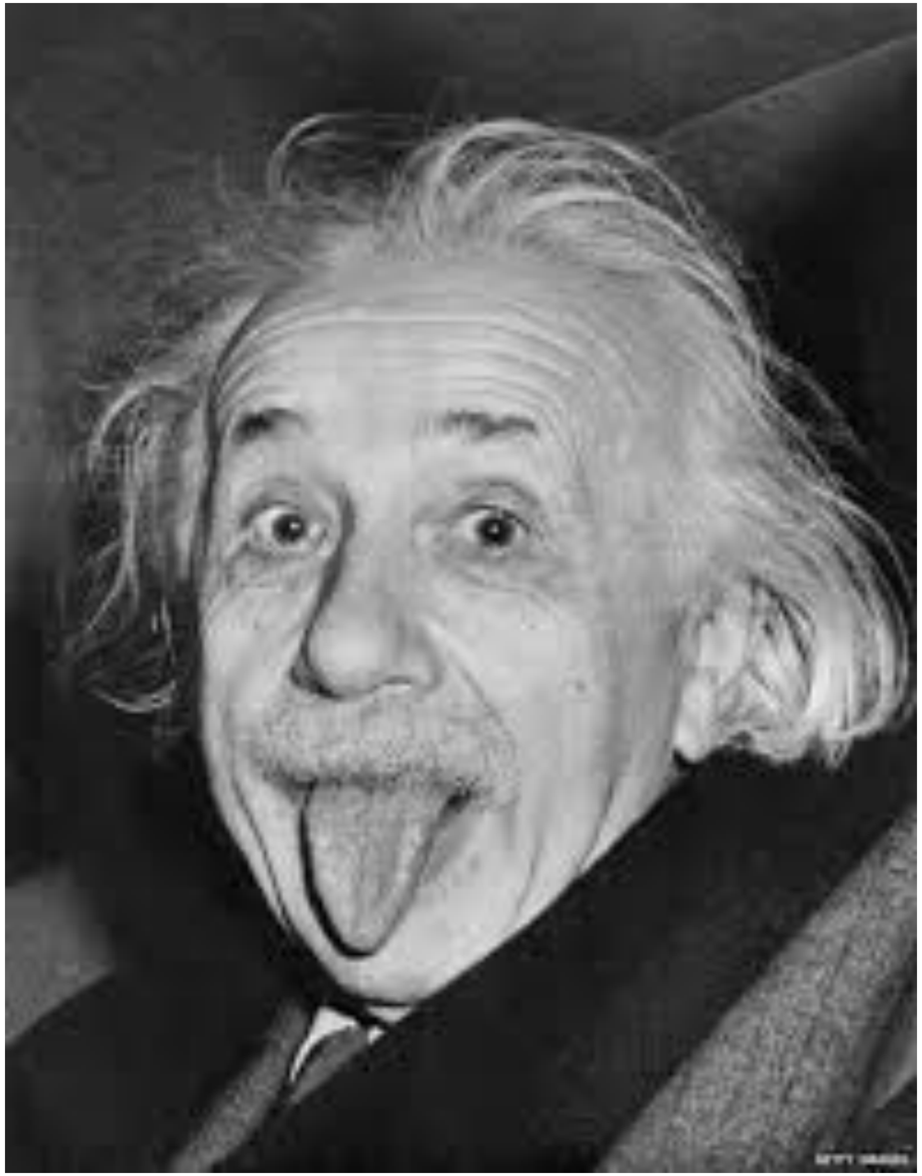


(a) When an object undergoes in-depth rotation, local features pooled over color domain preserves better alignment in final representation than spatial domain in a convolutional architecture.



(b) Overview of multi-scale and multi-domain pooling architecture


Physicists who use numbers
Physicists whom use letters
Physicians whom'st use tensor notation

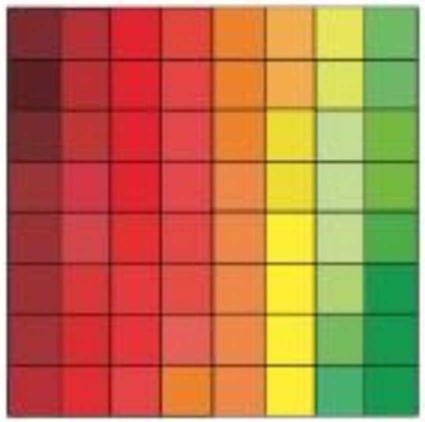


tensor = multidimensional array

vector

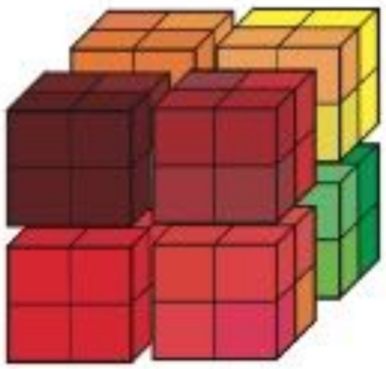
$\mathbf{v} \in \mathbb{R}^{64}$

matrix



$X \in \mathbb{R}^{8x8}$

tensor



$\mathcal{X} \in \mathbb{R}^{4x4x4}$

It is SHAPE that matters!

Layer Type	Configuration
DATA	input size: $28 \times 28 \times 1$
CONV	$k = 5, s = 1, p = 0, n = 20$
POOLING	MAX, $k = 2, s = 2, p = 0$
CONV	$k = 5, s = 1, p = 0, n = 50$
POOLING	MAX, $k = 2, s = 2, p = 0$
Dense	$n = 500$
RELU	
Dense	$n = 10$
LOSS	

LeNet!

Layer Type	Configuration
DATA	input size: $28 \times 28 \times 1$
CONV	$k = 5, s = 1, p = 0, n = 20$
POOLING	MAX, $k = 2, s = 2, p = 0$
CONV	$k = 5, s = 1, p = 0, n = 50$
POOLING	MAX, $k = 2, s = 2, p = 0$
Dense	$n = 500$
RELU	
Dense	$n = 10$
LOSS