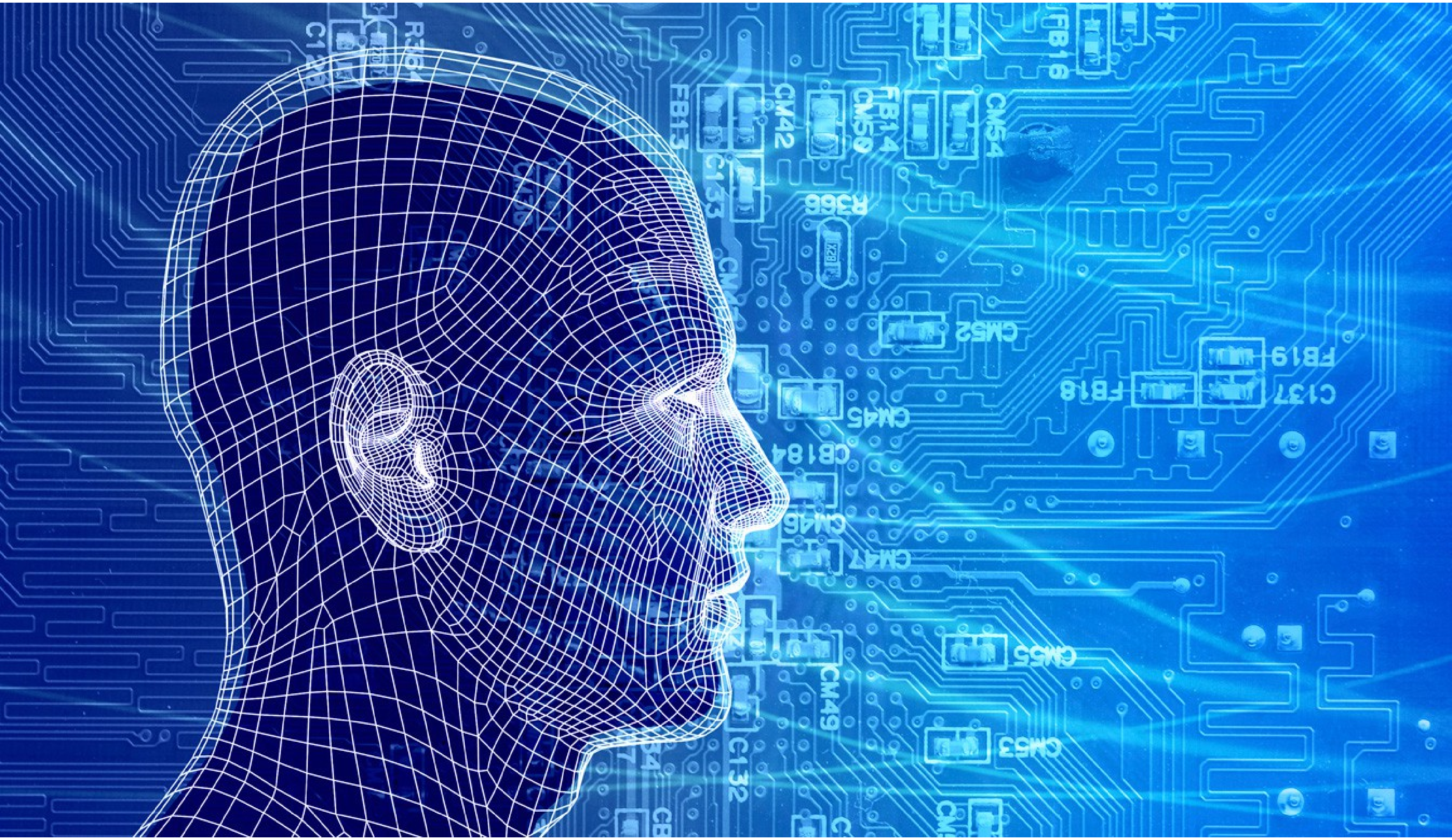
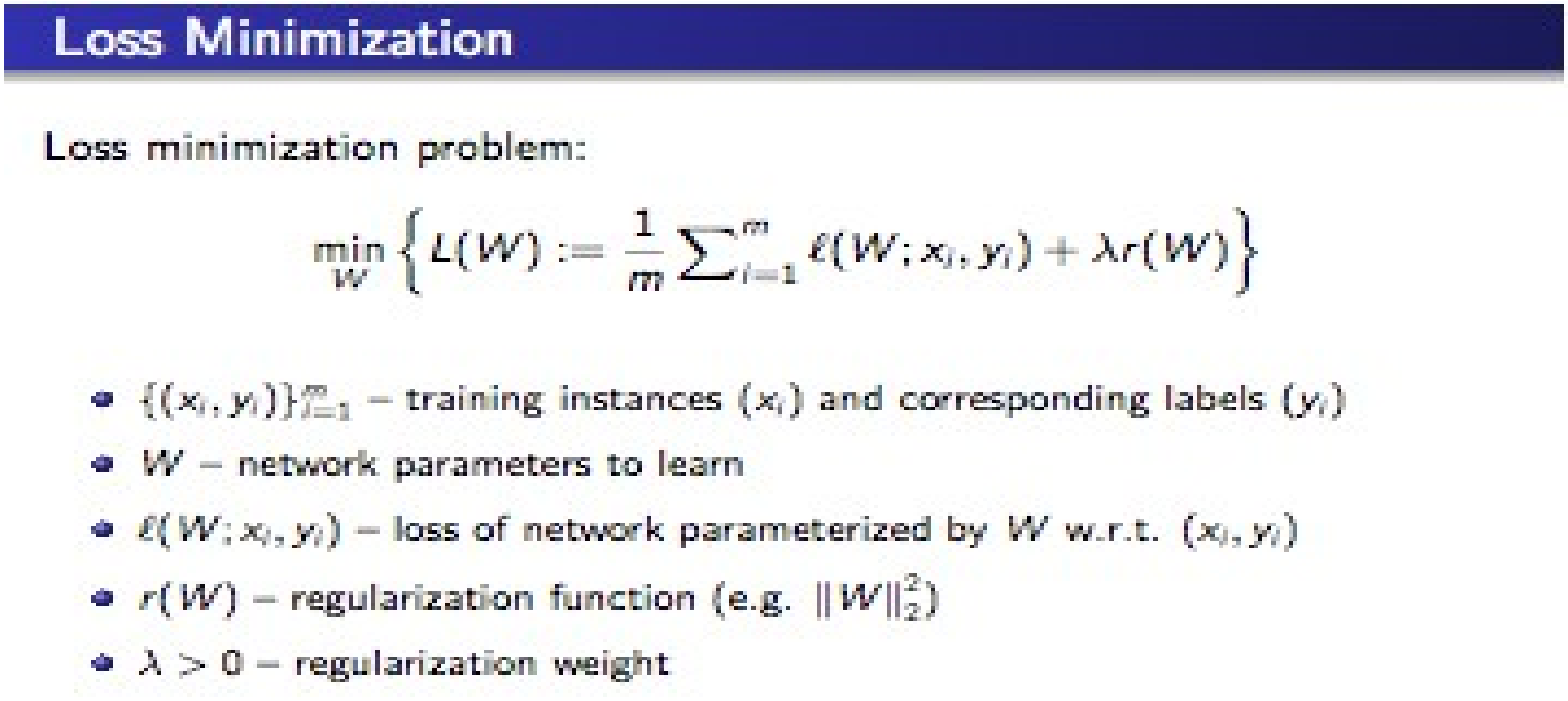
# Can machines think like human beings and beyond!



## Loss and Minimization Recap



### Loss Minimization

Loss minimization problem:

$$\min_{W}\left\{L(W) := \frac{1}{m}\sum_{i=1}^{m}\ell(W; x_i, y_i) + \lambda r(W)\right\}$$

- $\{(x_i, y_i)\}_{i=1}^{m}$ – training instances ($x_i$) and corresponding labels ($y_i$)
- $W$ – network parameters to learn
- $\ell(W; x_i, y_i)$ – loss of network parameterized by $W$ w.r.t. $(x_i, y_i)$
- $r(W)$ – regularization function (e.g. $\|W\|_2^2$)
- $\lambda > 0$ – regularization weight

## Large-Scale $\longrightarrow$ First-Order Stochastic Methods

Large-scale setting:

- Many network parameters (e.g. $dim(W) \sim 10^6$)
  $\Longrightarrow$ computing Hessian (second order derivatives) is expensive
- Many training examples (e.g. $m \sim 10^6$)
  $\Longrightarrow$ computing full objective at every iteration is expensive

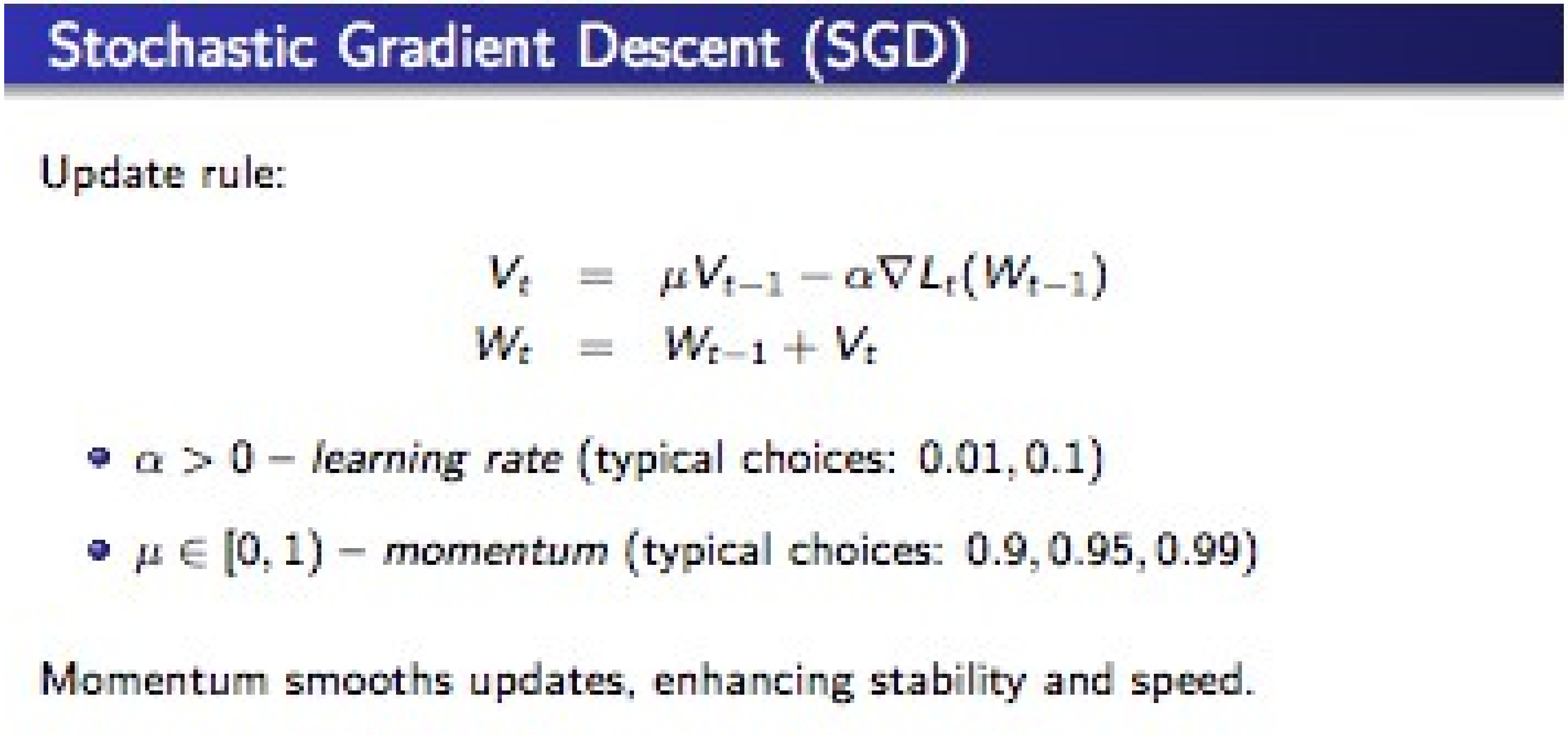
Optimization methods must be:

- **First-order** – update based on objective value and gradient only
- **Stochastic** – update based on subset of training examples:

$$L_t(W) := \frac{1}{b}\sum_{j=1}^{b} \ell(W; x_{i_j}, y_{i_j}) + \lambda r(W)$$

$\{(x_{i_j}, y_{i_j})\}_{j=1}^{b}$ – random *mini-batch* chosen at iteration $t$

## SGD



### Stochastic Gradient Descent (SGD)

Update rule:

$$\begin{aligned} V_t &= \mu V_{t-1} - \alpha \nabla L_t(W_{t-1}) \\ W_t &= W_{t-1} + V_t \end{aligned}$$

- $\alpha > 0$ – *learning rate* (typical choices: $0.01, 0.1$)
- $\mu \in [0, 1)$ – *momentum* (typical choices: $0.9, 0.95, 0.99$)

Momentum smooths updates, enhancing stability and speed.

## Nesterov's Accelerated Gradient (NAG)

Update rule:

$$\begin{aligned} V_t &= \mu V_{t-1} - \alpha \nabla L_t(W_{t-1} + \mu V_{t-1}) \\ W_t &= W_{t-1} + V_t \end{aligned}$$

Only difference from SGD is partial update $(+\mu V_t)$ in gradient computation. May increase stability and speed in ill-conditioned problems[1].

## Adaptive Gradient (AdaGrad)

Update rule:

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{\sum_{t'=1}^{t} \nabla L_{t'}(W_{t'-1})^2}}$$

Learning rate adapted per coordinate:

- Highly varying coordinate $\longrightarrow$ suppress
- Rarely varying coordinate $\longrightarrow$ enhance

Disadvantage in non-stationary settings:
All gradients (recent and old) weighted equally

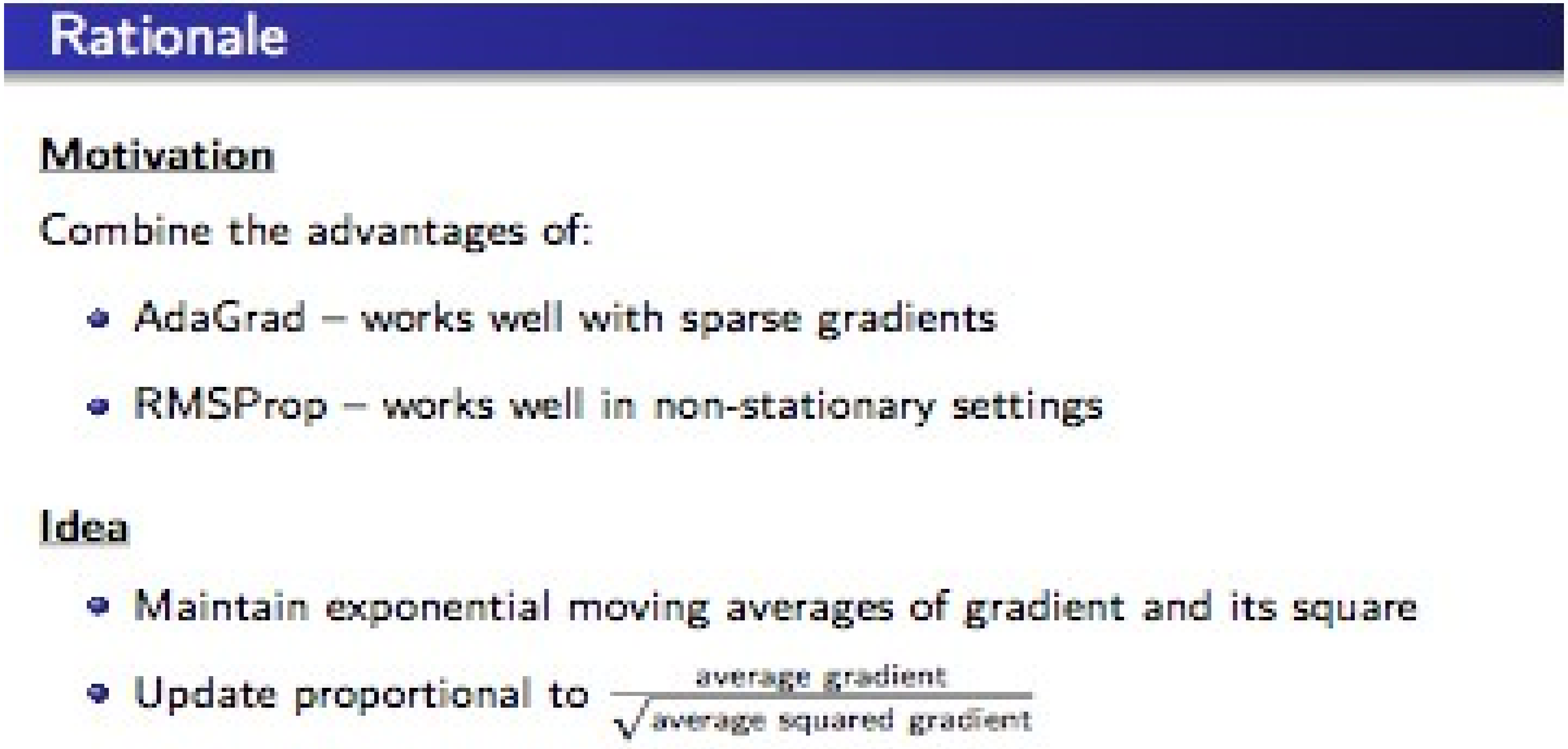
## Root Mean Square Propagation (RMSProp)

Update rule:

$$\begin{aligned} R_t &= \gamma R_{t-1} + (1-\gamma)\nabla L_t(W_{t-1})^2 \\ W_t &= W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{R_t}} \end{aligned}$$

Similar to AdaGrad but with an exponential moving average controlled by $\gamma \in [0, 1)$ (smaller $\gamma \implies$ more emphasis on recent gradients).

# Adam- Adaptive Moment Estimation



## Rationale

**Motivation**

Combine the advantages of:

- AdaGrad – works well with sparse gradients
- RMSProp – works well in non-stationary settings

**Idea**

- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

## Algorithm

**Adam**

```
$M_0 = \mathbf{0}, R_0 = \mathbf{0}$    (Initialization)
For $t = 1, \ldots, T$:
    $M_t = \beta_1 M_{t-1} + (1 - \beta_1)\nabla L_t(W_{t-1})$    (1st moment estimate)
    $R_t = \beta_2 R_{t-1} + (1 - \beta_2)\nabla L_t(W_{t-1})^2$    (2nd moment estimate)
    $\hat{M}_t = M_t / (1 - (\beta_1)^t)$    (1st moment bias correction)
    $\hat{R}_t = R_t / (1 - (\beta_2)^t)$    (2nd moment bias correction)
    $W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t + \epsilon}}$    (Update)
Return $W_T$
```

Hyper-parameters:

- $\alpha > 0$ – learning rate (typical choice: 0.001)
- $\beta_1 \in [0, 1)$ – 1st moment decay rate (typical choice: 0.9)
- $\beta_2 \in [0, 1)$ – 2nd moment decay rate (typical choice: 0.999)
- $\epsilon > 0$ – numerical term (typical choice: $10^{-8}$)

## Parameter Updates

Adam's step at iteration $t$ (assuming $\epsilon = 0$):

$$\Delta_t = -\alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t}}$$

Properties:

- **Scale-invariance:**

$$\begin{aligned} L(W) \to c \cdot L(W) \quad &\Longrightarrow \quad \hat{M}_t \to c \cdot \hat{M}_t \;\wedge\; \hat{R}_t \to c^2 \cdot \hat{R}_t \\ &\Longrightarrow \quad \Delta_t \text{ does not change} \end{aligned}$$

- **Bounded norm:**

$$\|\Delta_t\|_\infty \leq \begin{cases} \alpha \cdot (1-\beta_1)/\sqrt{1-\beta_2} & ,(1-\beta_1) > \sqrt{1-\beta_2} \\ \alpha & ,\text{otherwise} \end{cases}$$

## Convergence in Online Convex Regime

*Regret* at iteration $T$ :

$$R(T) := \sum_{t=1}^{T} [L_t(W_t) - L_t(W^*)]$$

where:

$$W^* := \underset{W}{\operatorname{argmin}} \sum_{t=1}^{T} L_t(W)$$

In convex regime, Adam gives regret bound comparable to best known:

**Theorem**

*If all batch objectives $L_t(W)$ are convex and have bounded gradients, and all points $W_t$ generated by Adam are within bounded distance from each other, then for every $T \in \mathbb{N}$:*

$$\frac{R(T)}{T} = O\left(\frac{1}{\sqrt{T}}\right)$$

## Relation to SGD

Setting $\beta_2 = 1$, Adam's update rule may be written as:

$$\begin{aligned} M_t &= \mu M_{t-1} - \eta \nabla L_t(W_{t-1}) \\ W_t &= W_{t-1} + M_t \end{aligned}$$

where:

$$\mu := \frac{\beta_1(1-(\beta_1)^{t-1})}{1-(\beta_1)^t} \qquad \eta := \frac{\alpha(1-\beta_1)}{(1-(\beta_1)^t)\sqrt{\epsilon}}$$

<u>Conclusion</u>

Disabling 2nd moment estimation ($\beta_2 = 1$) reduces Adam to SGD with:

- Learning rate descending towards $\alpha(1-\beta_1)/\sqrt{\epsilon}$
- Momentum ascending towards $\beta_1$

## Relation to AdaGrad

Setting $\beta_1 = 0$ and $\beta_2 \to 1^-$ (and assuming $\epsilon = 0$), Adam's update rule may be written as:

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{t^{1/2}\sqrt{\sum_{i=1}^{t} \nabla L_t(W_{t-1})^2}}$$

<u>**Conclusion**</u>

In the limit $\beta_2 \to 1^-$, with $\beta_1 = 0$, Adam reduces to AdaGrad with annealing learning rate $\alpha \cdot t^{-1/2}$.

## Relation to RMSProp

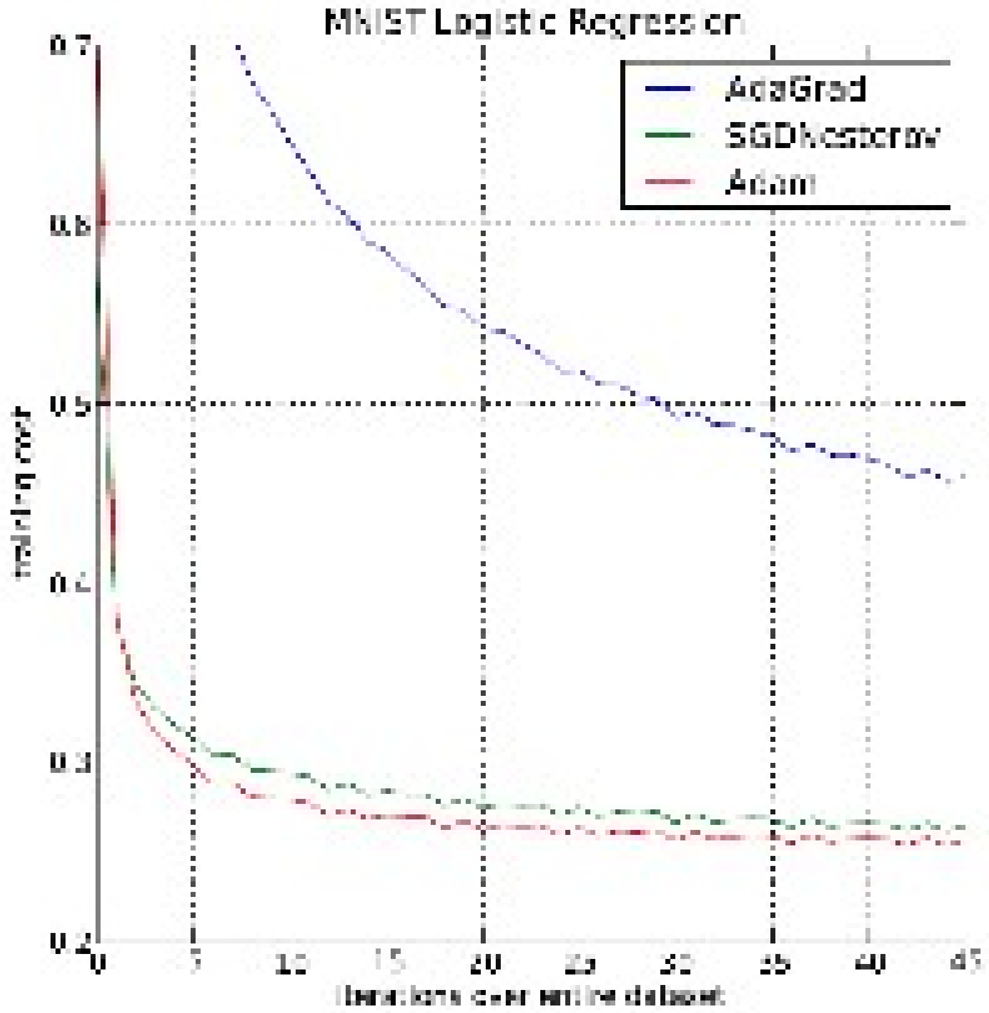
RMSProp with momentum is the method most closely related to Adam.

Main differences:

- RMSProp rescales gradient and then applies momentum, Adam first applies momentum (moving average) and then rescales.
- RMSProp lacks bias correction, often leading to large stepsizes in early stages of run (especially when $\beta_2$ is close to 1).

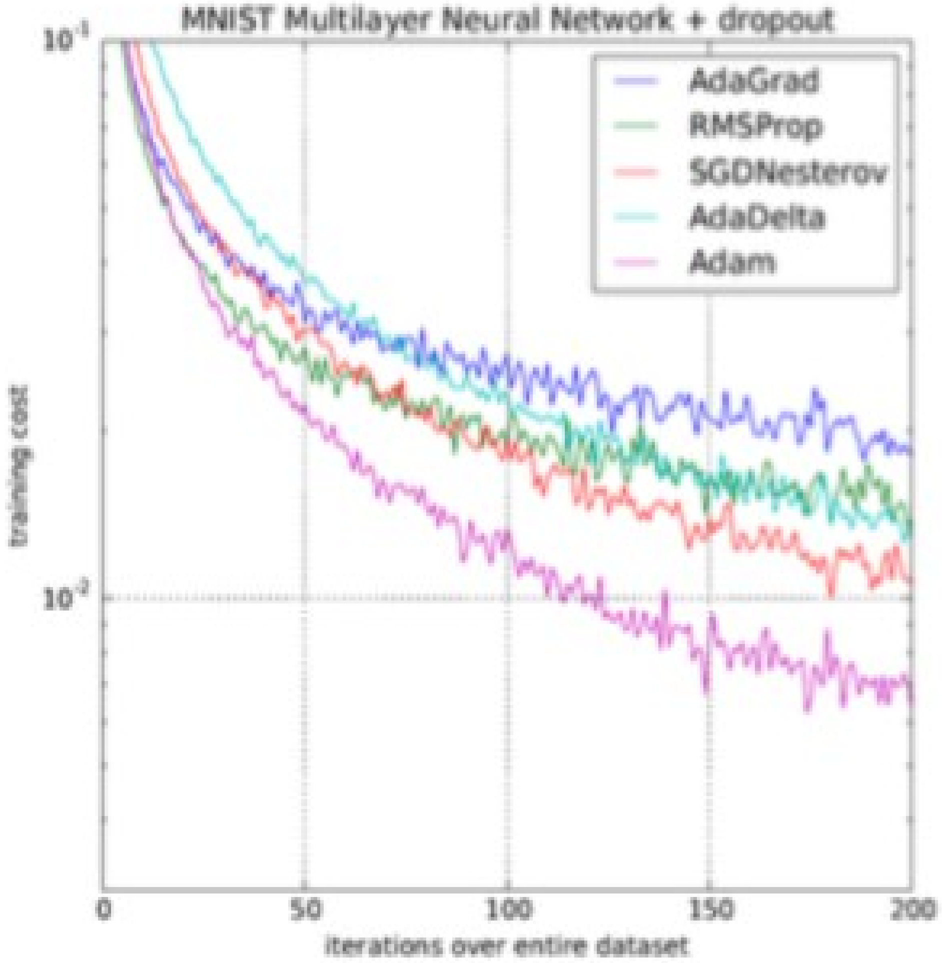
## Logistic Regression on MNIST

$L^2$-regularized logistic regression applied directly to image pixels:



## Multi-Layer Neural Networks (Fully-Connected) on MNIST

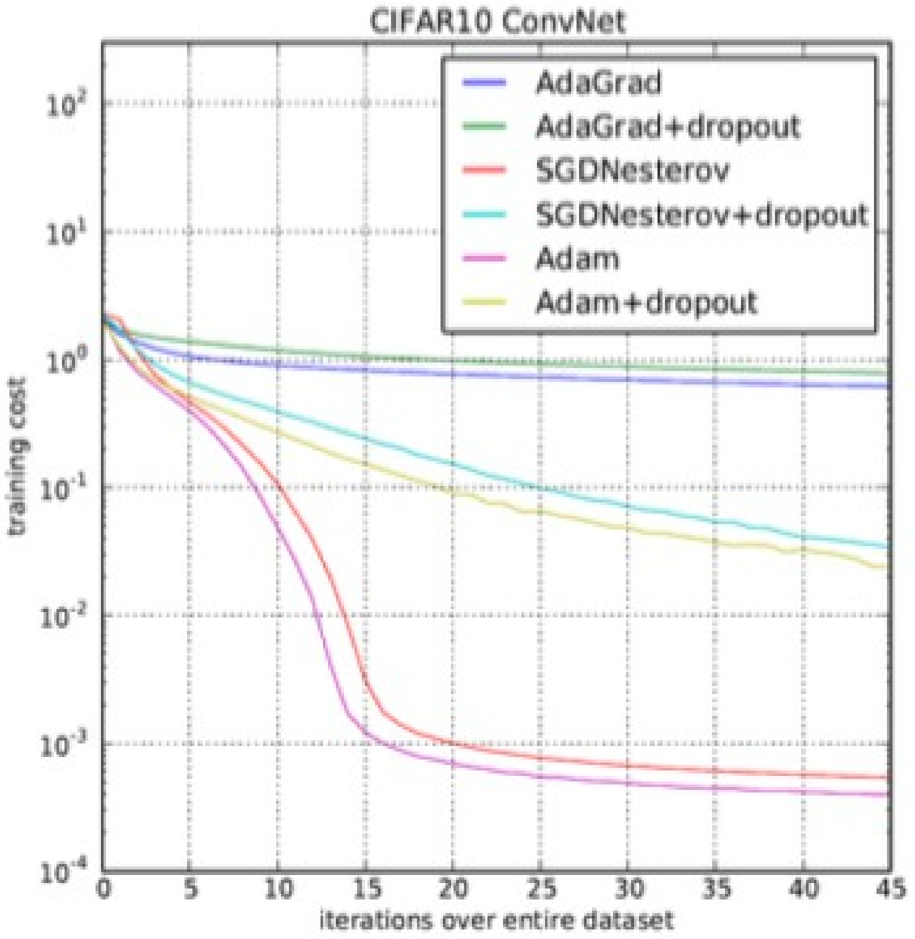
2 hidden layers, 1000 units each, ReLU activation, dropout regularization:



## Convolutional Neural Networks on CIFAR-10

Network architecture:

conv-5x5@64 $\rightarrow$ pool-3x3(stride-2) $\rightarrow$ conv-5x5@64 $\rightarrow$ pool-3x3(stride-2) $\rightarrow$ conv-5x5@128 $\rightarrow$ pool-3x3(stride-2) $\rightarrow$ dense@1000 $\rightarrow$ dense@10:

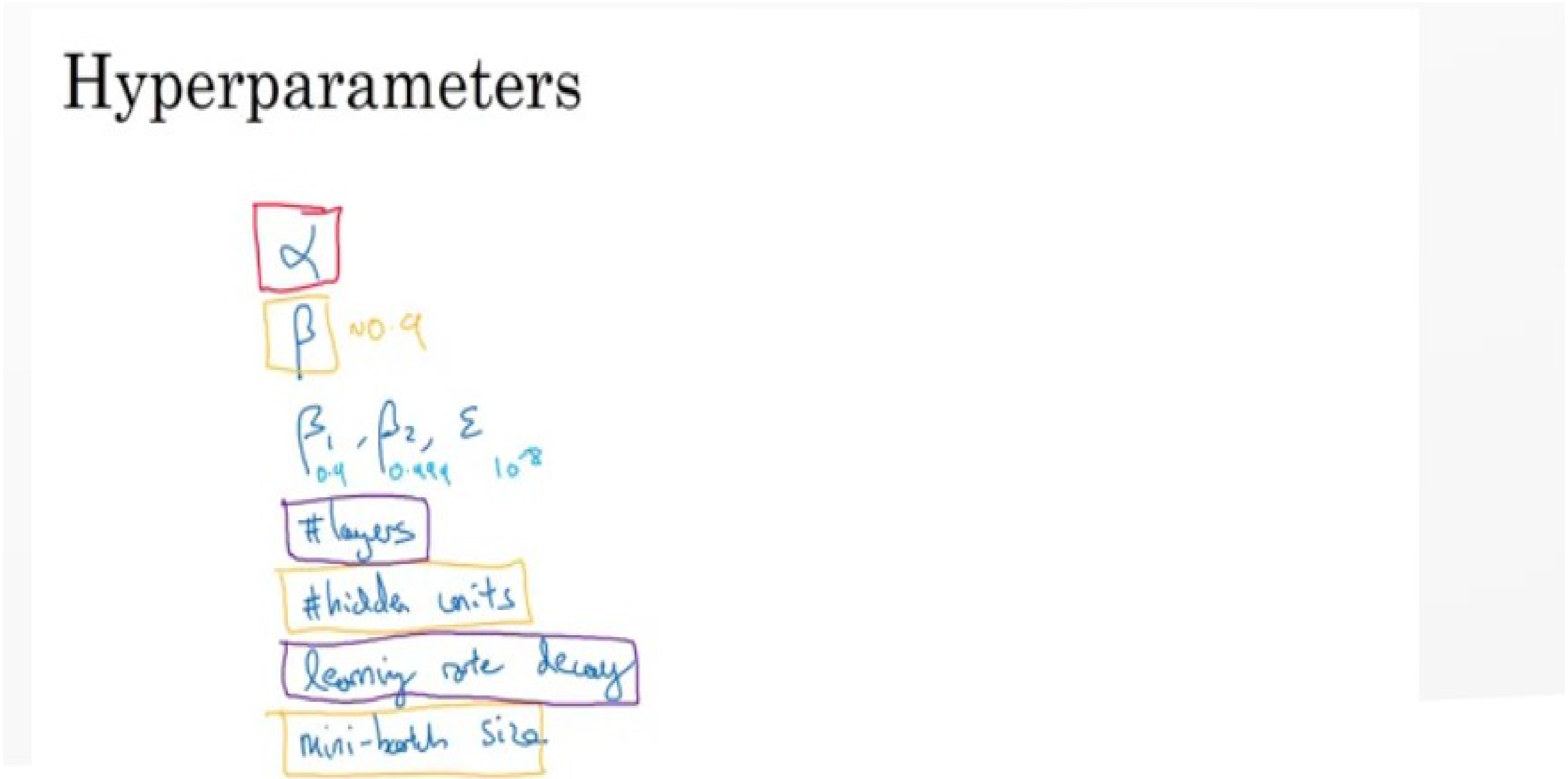


## Conclusion

**Adam:**

- Efficient first-order stochastic optimization method
- Combines the advantages of:
  - AdaGrad – works well with sparse gradients
  - RMSProp – deals with non-stationary objectives
- Parameter updates:
  - Have bounded norm
  - Are scale-invariant
- *Widely used in deep learning community (e.g. Google DeepMind)*

# Hyper parameters



Hyperparameters

$\boxed{\alpha}$

$\boxed{\beta}$ ~0.9

$\underset{0.9}{\beta_1}, \underset{0.999}{\beta_2}, \underset{10^{-8}}{\varepsilon}$

$\boxed{\text{\# layers}}$

$\boxed{\text{\# hidden units}}$

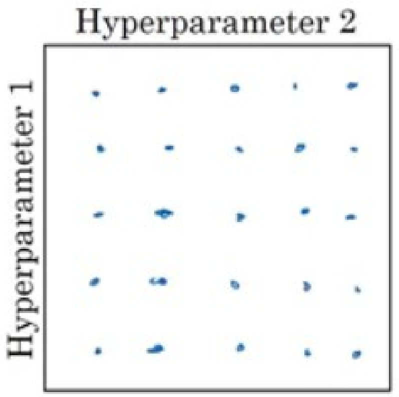
$\boxed{\text{learning rate decay}}$

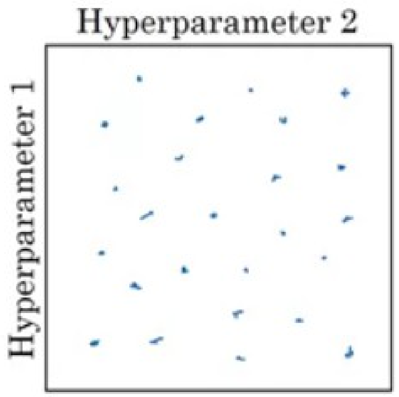
$\boxed{\text{mini-batch size}}$

# Grid Vs Random approach

Tuning process

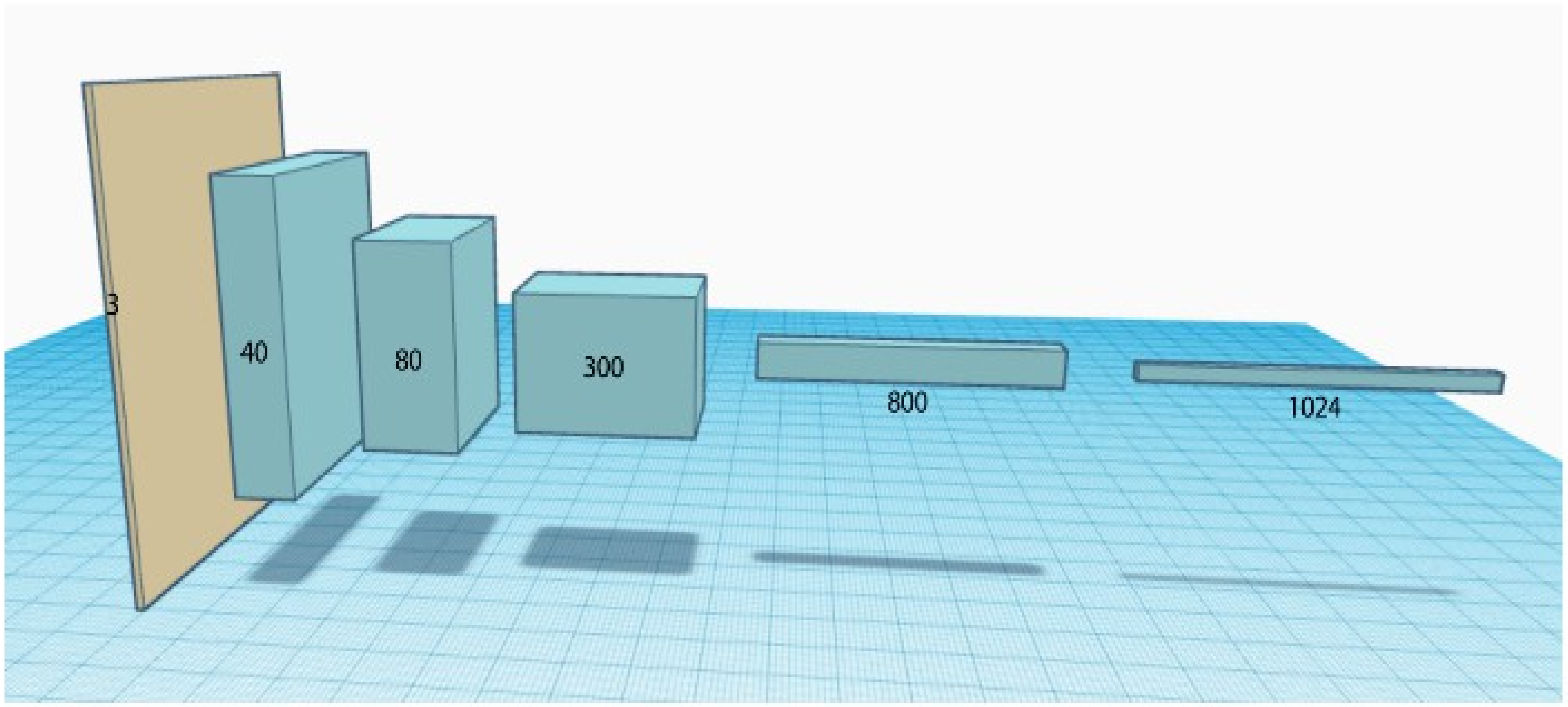
Try random values: Don't use a grid





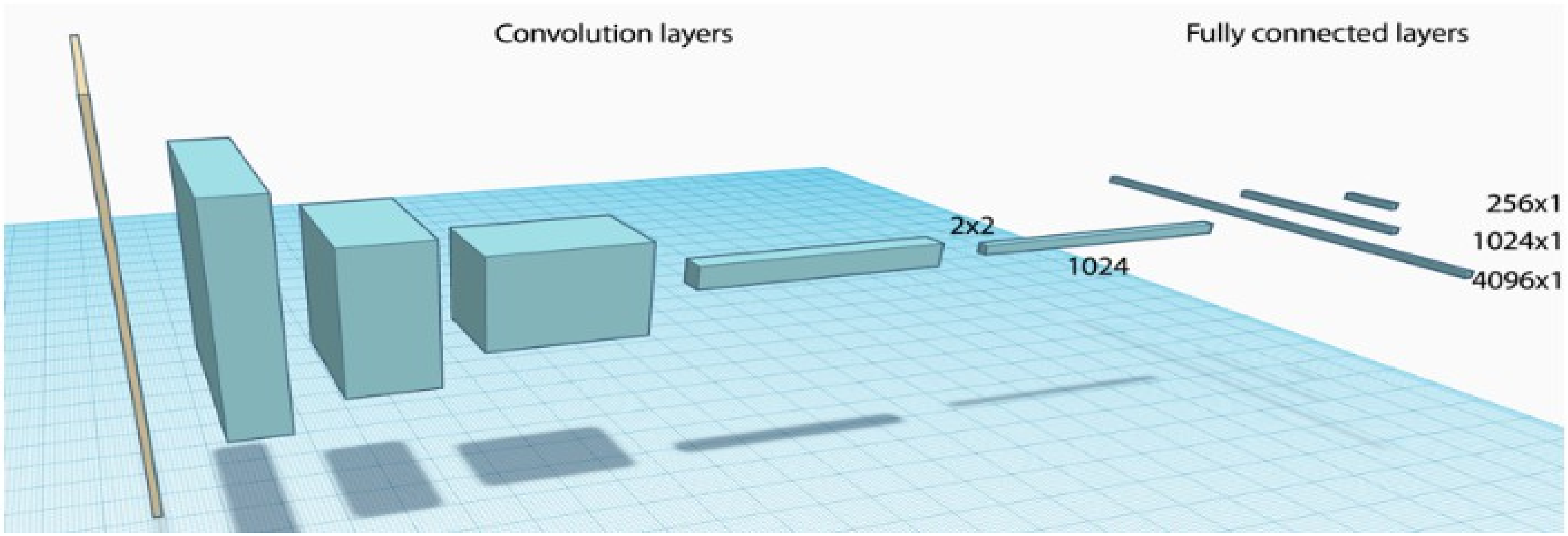
## Increase model capacity

To increase the capacity, we add layers and nodes to a deep network (DN) gradually. **Deeper layers** produce more complex models. We also reduce filter sizes. Smaller filters (3x3 or 5x5) usually perform better than larger filters.



The tuning process is more **empirical** than theoretical. We add layers and nodes gradually with the intention to overfit the model since we can tone it down with regularizations. We repeat the iterations until the accuracy improvement is diminishing and no longer justify the drop in the training and computation performance.

However, GPUs do not page out memory. As of early 2018, the high-end NVIDIA GeForce GTX 1080 TI has 11GB memory. The maximum number of hidden nodes between two affine layers is restricted by the memory size.



## Auto encoders

Thank You!