Map of RL Algorithms

Model Based

Model-Free

Learning Q

Learning π

On Policy

SARSA

Policy Grad

MLMBTRL

(Learn T, R)

Q-learning

Off Policy

Tabular

This Time

Challenges in Reinforcement Learning:

- Exploration vs Exploitation
- Credit Assignment
- Generalization

Function Approximation

Function Approximation

$$y \approx f_\theta(x)$$

Function Approximation

input

$$y \approx f_\theta(x)$$

Function Approximation

input

$$y \approx f_\theta(x)$$

output

Function Approximation

input

$$y \approx f_\theta(x)$$

output

parameters

Function Approximation

input

$$y \approx f_\theta(x)$$

output parameters

Previously, Linear:

$$f_\theta(x) = \theta^\top \beta(x)$$

Function Approximation

input

$$y \approx f_\theta(x)$$

output parameters

Previously, Linear:

$$f_\theta(x) = \theta^\top \beta(x)$$

weights features

Function Approximation

input

$$y \approx f_\theta(x)$$

output parameters

Previously, Linear:

$$f_\theta(x) = \theta^\top \beta(x)$$

weights features

e.g. $\beta_i(x) = \sin(i\,\pi\,x)$

~~AI = Neural Nets~~

π_θ

gathering data

Q_θ

~~Neural Nets are just another function approximator~~

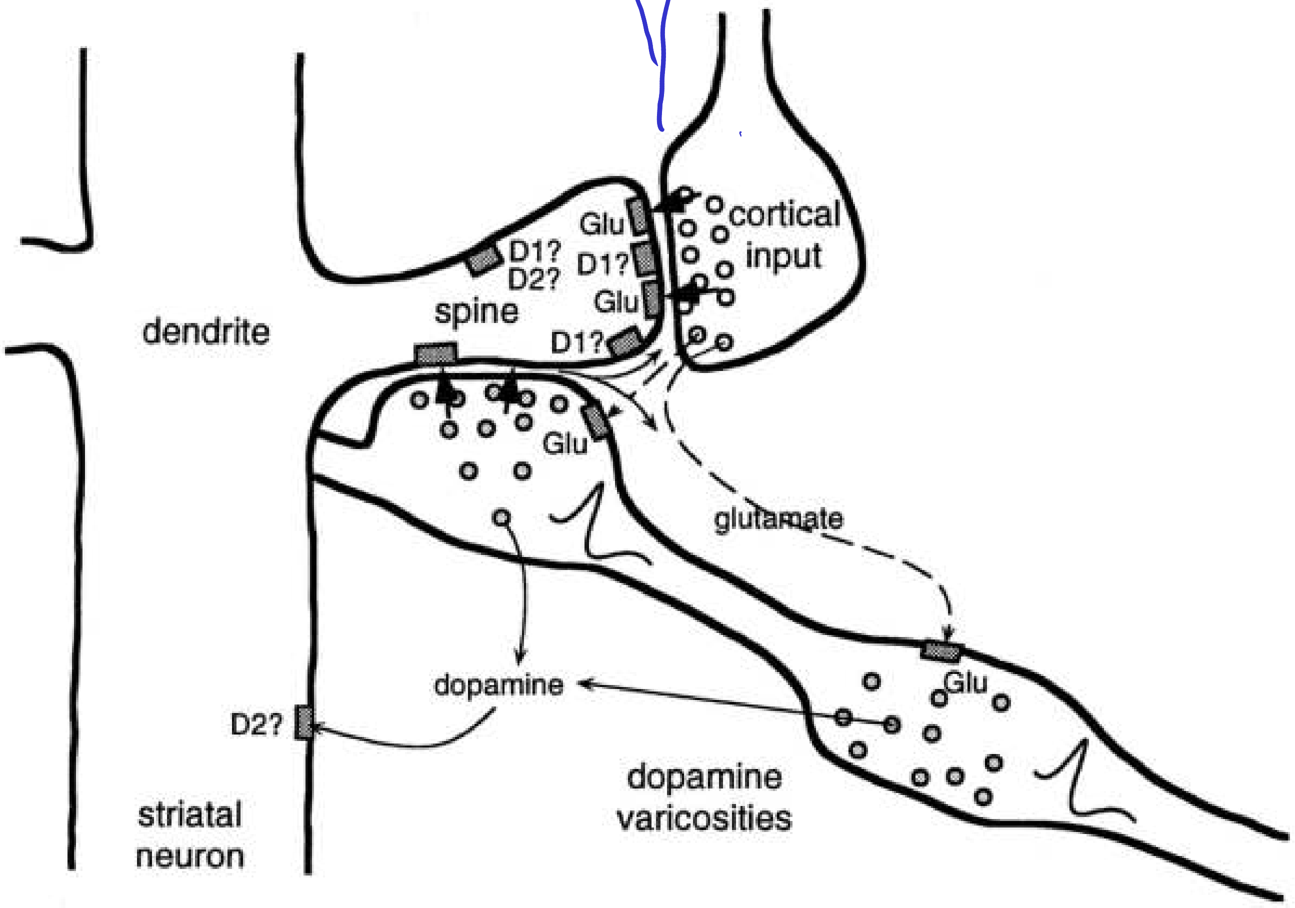
Neural Network

Neural Network

$$h(x) = \sigma(Wx + b)$$

Neural Network

$$h(x) = \sigma(Wx + b)$$



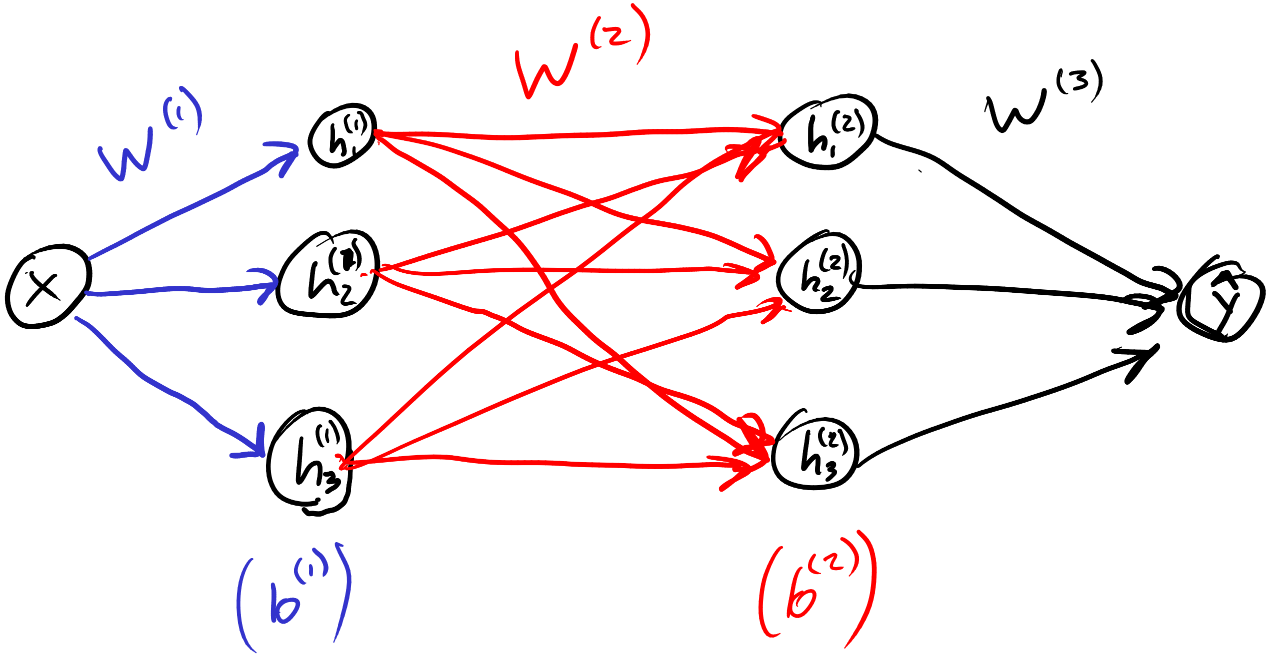
Neural Network

Neural Network

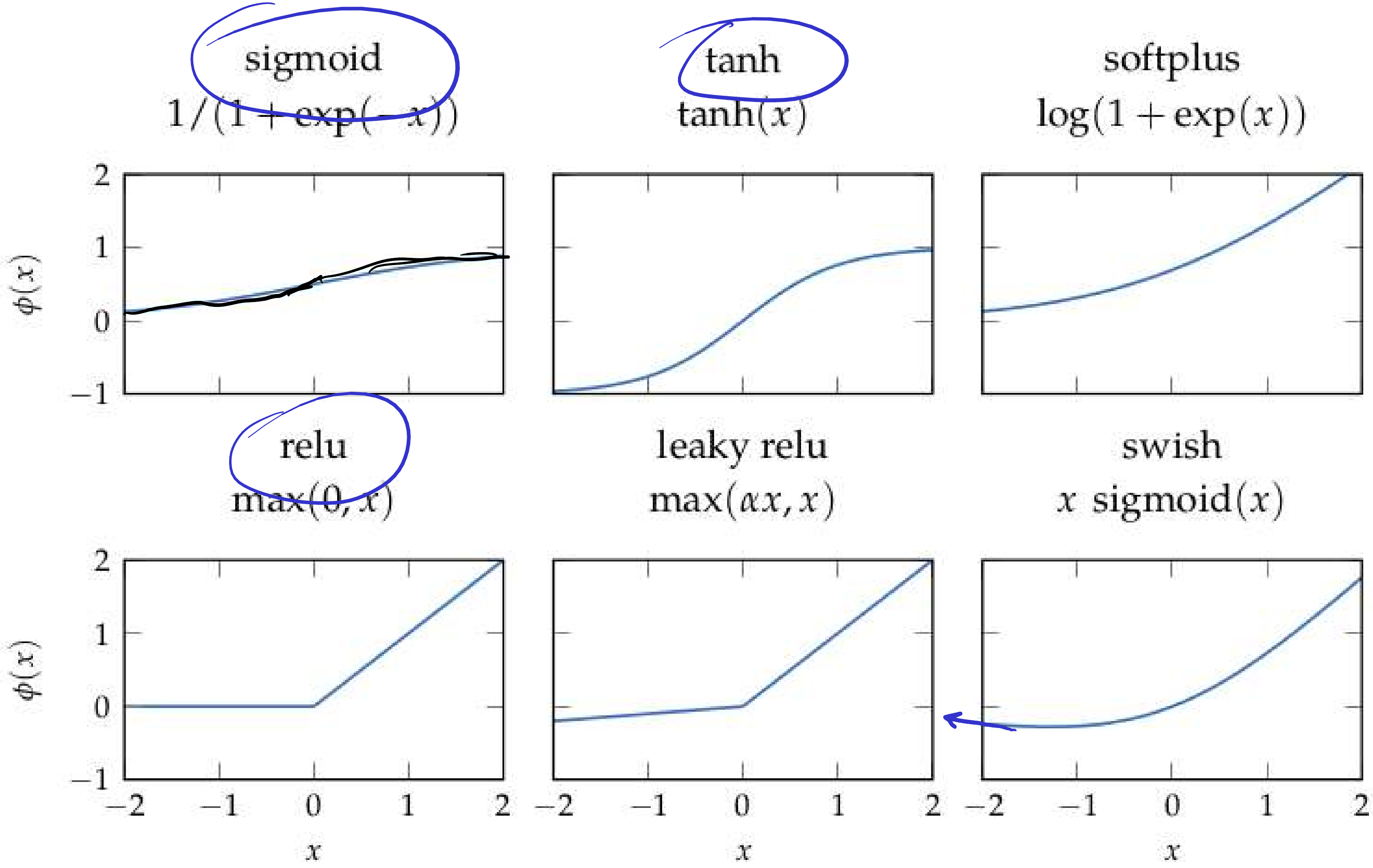
$$h(x) = \sigma(Wx + b)$$

$$f_\theta(x) = h^{(2)}(h^{(1)}(x)) = W^{3}\sigma^{(2)}(W^{(2)}\sigma^{(1)}(W^{(1)}x + b^{(1)}) + b^{(2)})$$

$$\theta = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$$

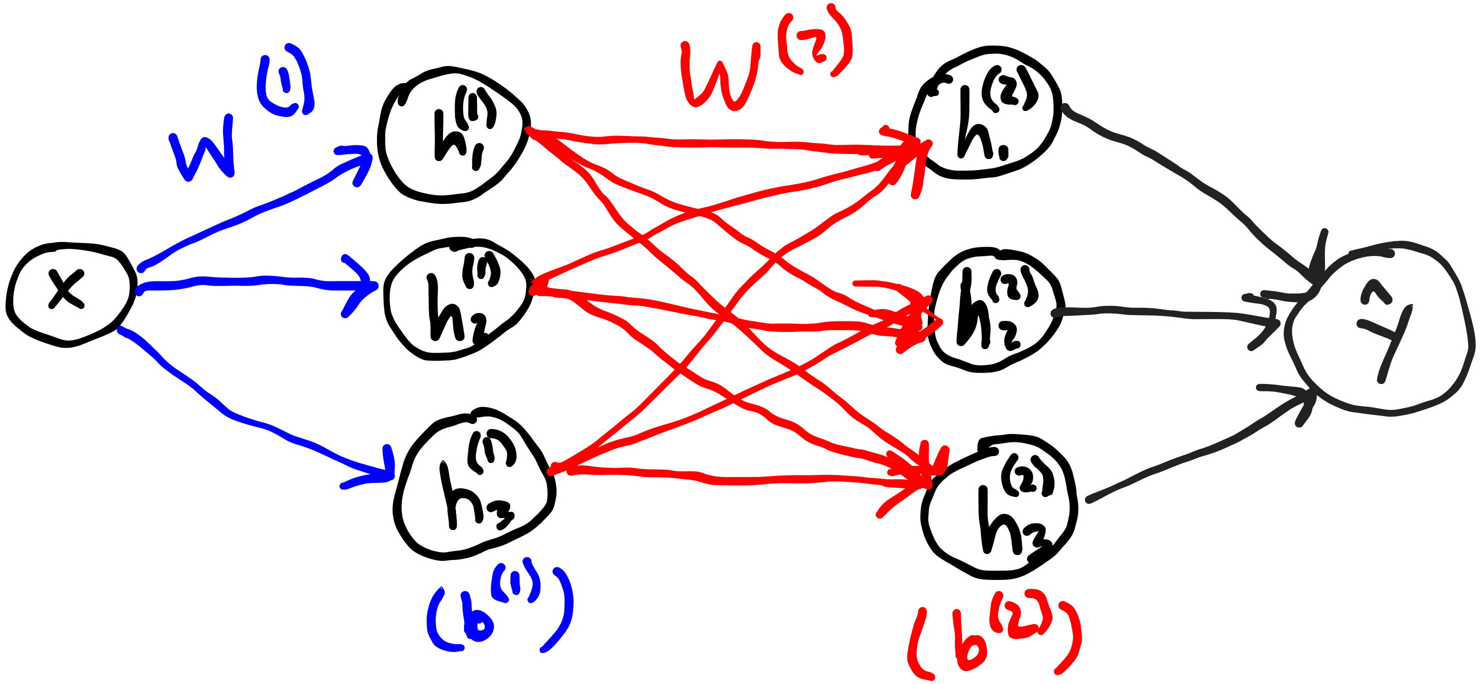


Nonlinearities

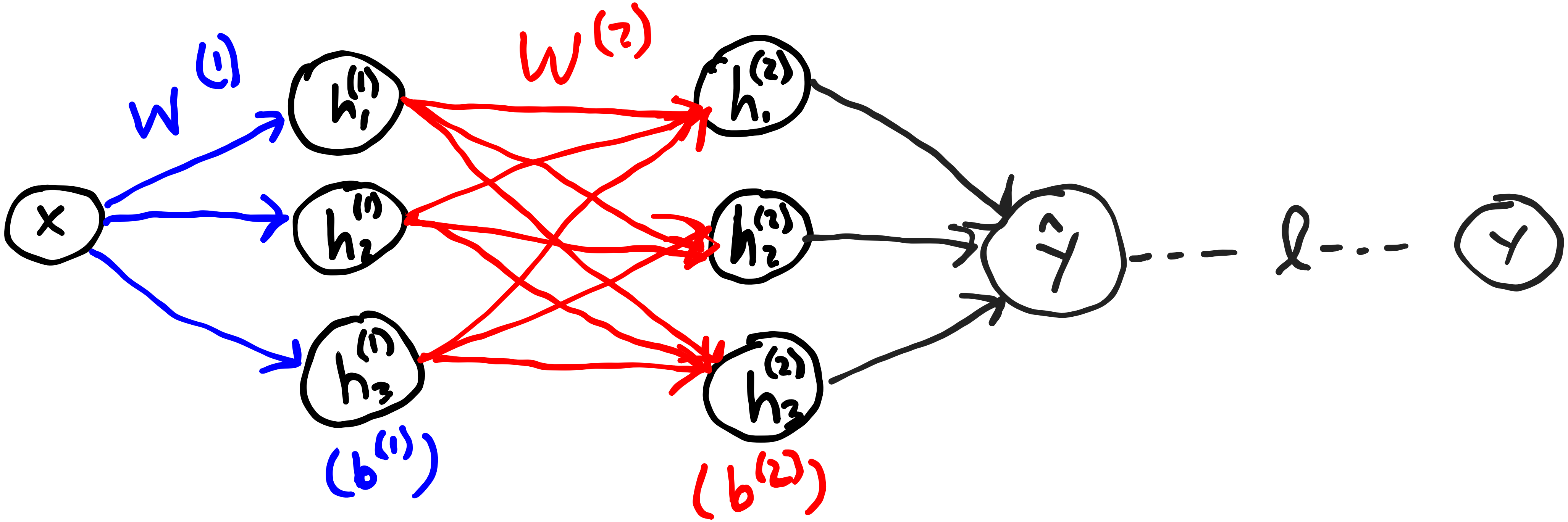


Training

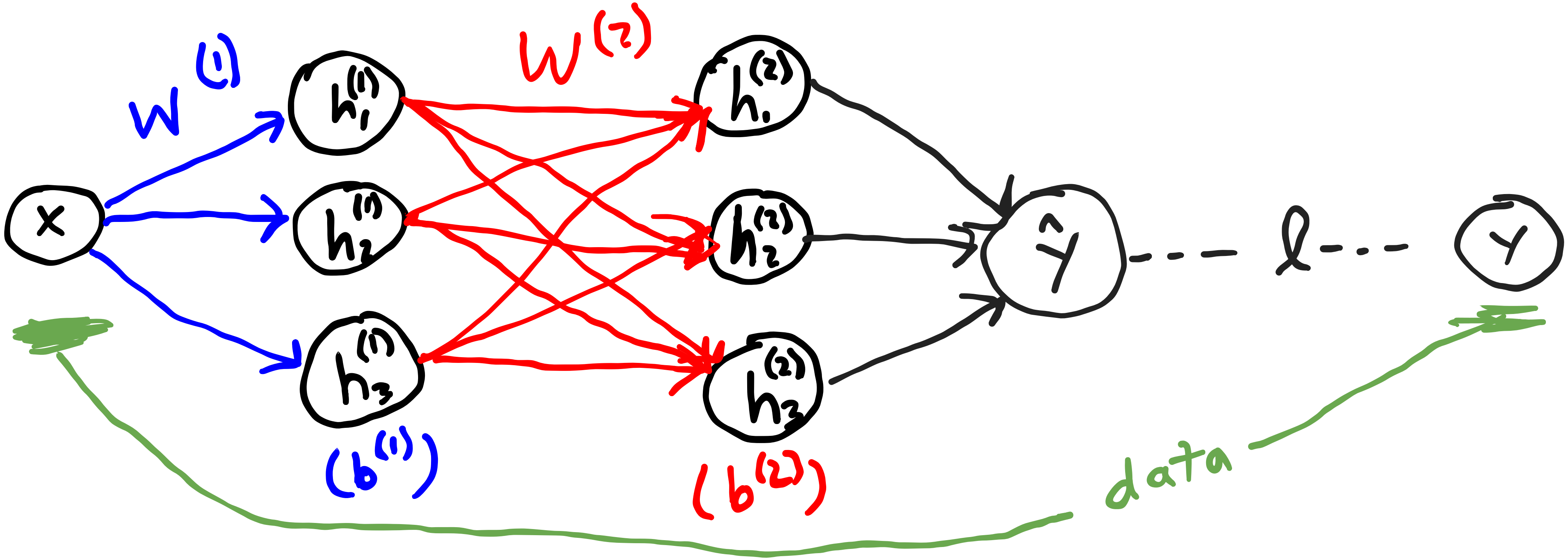
Training



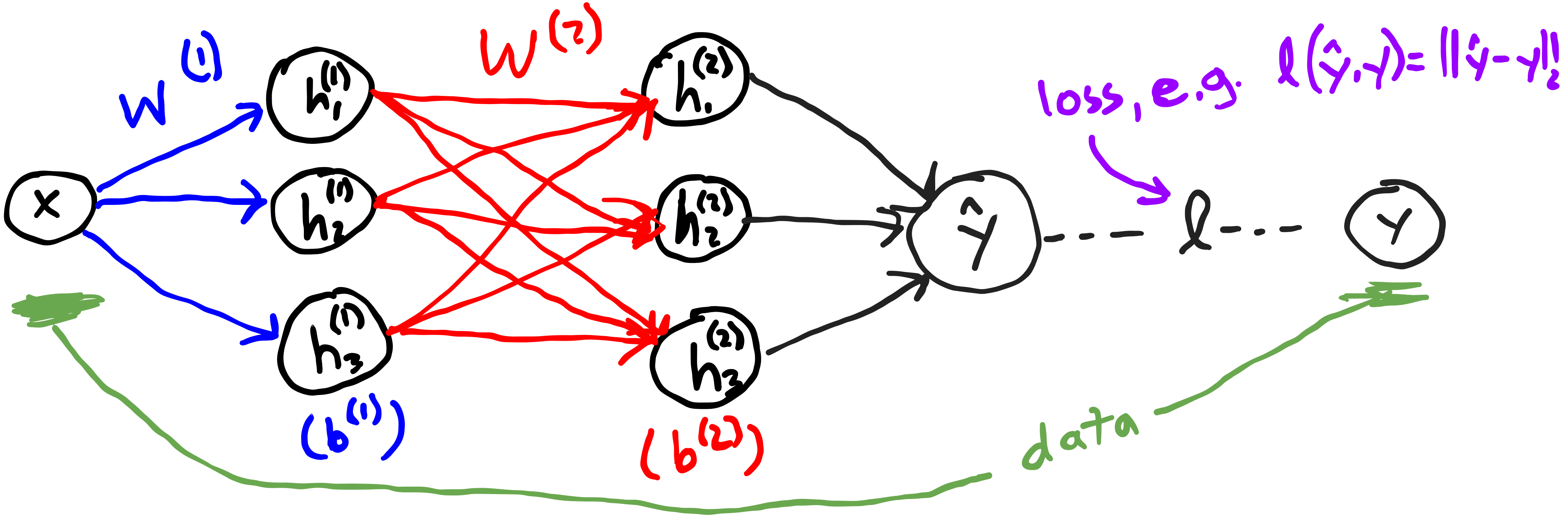
Training



Training

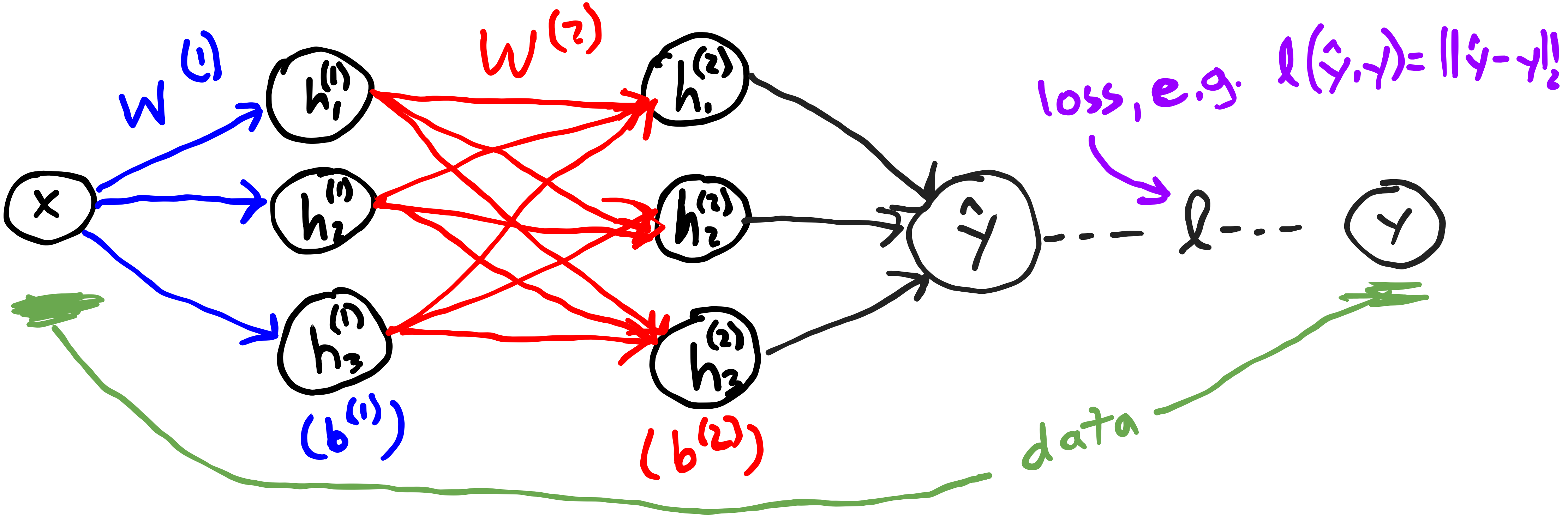


Training



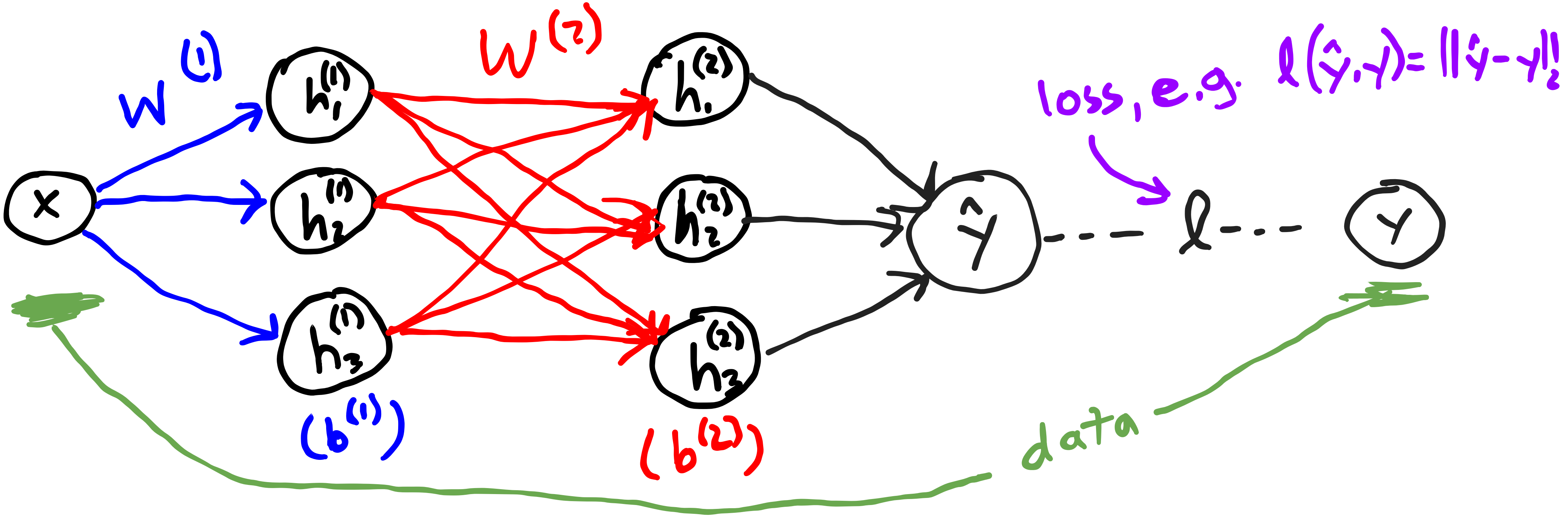
$W^{(1)}$
$W^{(2)}$
x
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_1^{(2)}$
$h_2^{(2)}$
$h_3^{(2)}$
$\hat{y}$
ℓ
y
$(b^{(1)})$
$(b^{(2)})$
loss, e.g. $\ell(\hat{y}, y) = \|\hat{y} - y\|_2^2$
data

Training



$$\theta^* = \arg\min_\theta \sum_{(x,y)\in\mathcal{D}} l(f_\theta(x), y)$$

Training



$$\theta^* = \arg\min_\theta \sum_{(x,y)\in\mathcal{D}} l(f_\theta(x), y)$$

Stochastic Gradient Descent: $\theta \leftarrow \theta - \alpha \nabla_\theta l(f_\theta(x), y)$

Chain Rule

$f \cdot g \cdot h$ $\quad f(g(h(x)))$

$$\left.\frac{\partial f(g(h(x)))}{\partial x}\right|_x = \left.\frac{\partial f(g(h))}{\partial h}\right|_h \left.\frac{\partial h(x)}{\partial x}\right|_x = \left.\frac{\partial f(g)}{\partial g}\right|_g \left.\frac{\partial g(h)}{\partial h}\right|_h \left.\frac{\partial h(x)}{\partial x}\right|_x$$

$$\hat{y} = W^{(2)} \sigma\left(W^{(1)} x + b^{(1)}\right) + b^{(2)}$$

$$\ell(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\frac{\partial \ell}{\partial \hat{y}_i} = -\frac{1}{n} 2 (y_i - \hat{y}_i)$$

$$\frac{\partial \ell}{\partial W^{(2)}} = \frac{\partial \ell}{\partial \hat{y}} \left(\frac{\partial \hat{y}}{\partial W^2}\right)^T = \frac{\partial \ell}{\partial \hat{y}} \sigma\left(W^{(1)} x + b^{(1)}\right)^T$$

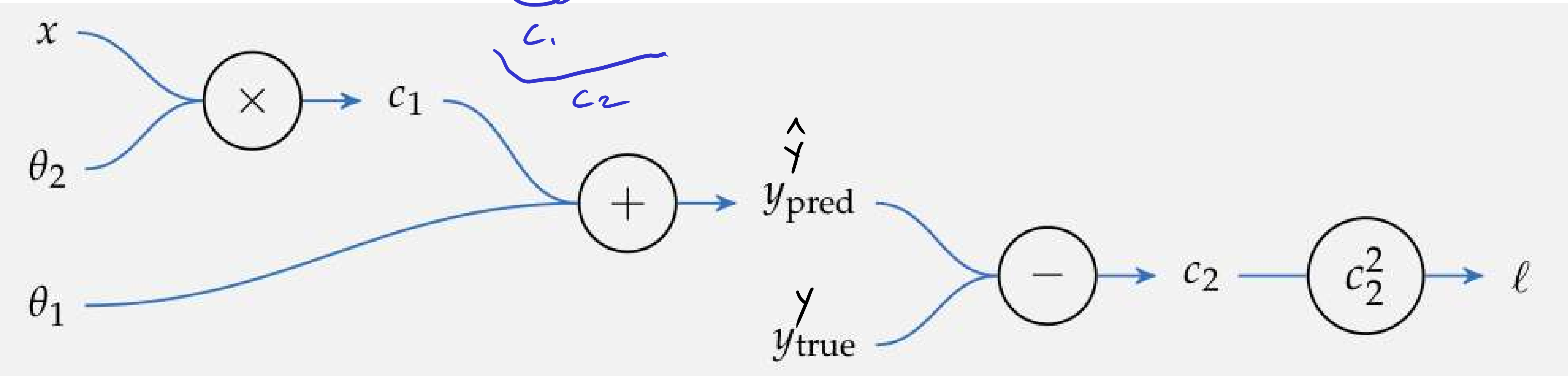
$$W^{(2)} \leftarrow W^{(2)} - \alpha \frac{\partial \ell}{\partial W^{(2)}}$$

Backprop

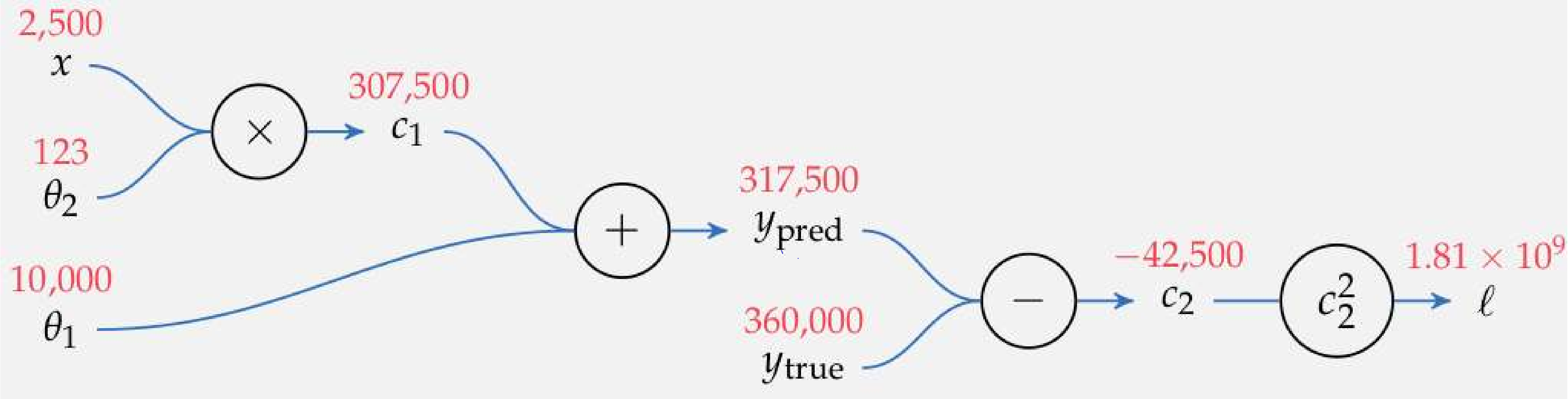
Backprop

$$\underbrace{\underbrace{\theta_2 x}_{c_1} + \theta}_{c_2}$$

$$\ell(y,\hat{y}) = (y-\hat{y})^2$$

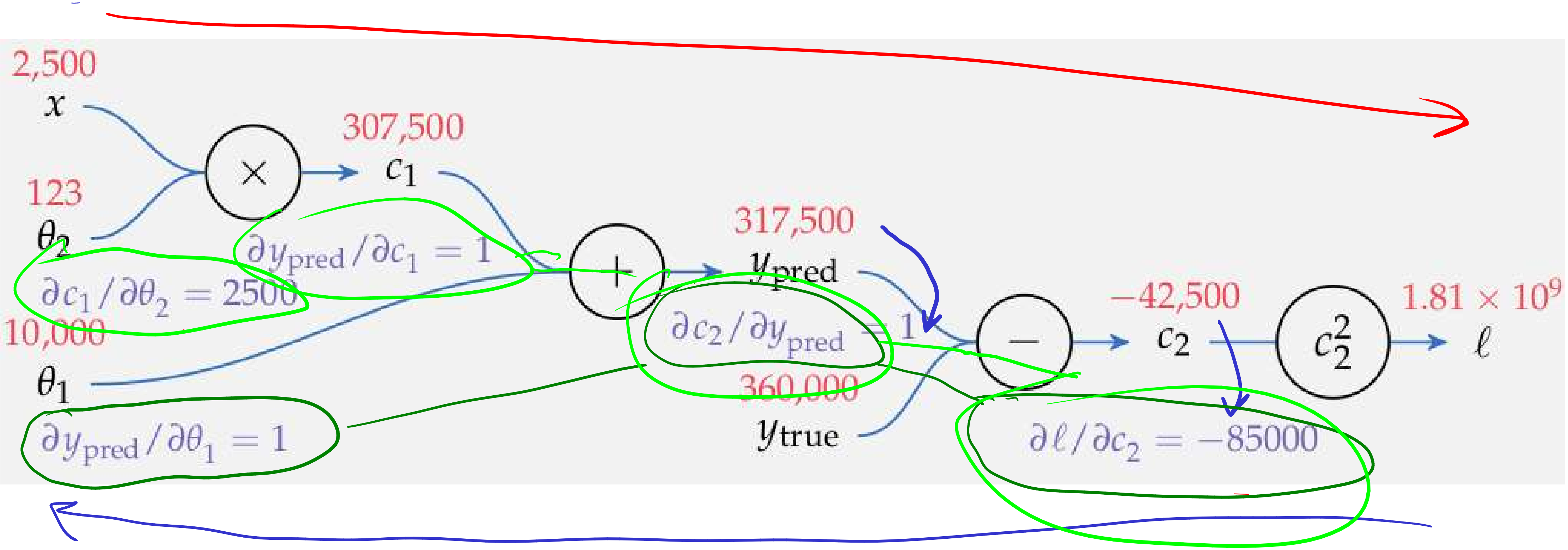


Backprop



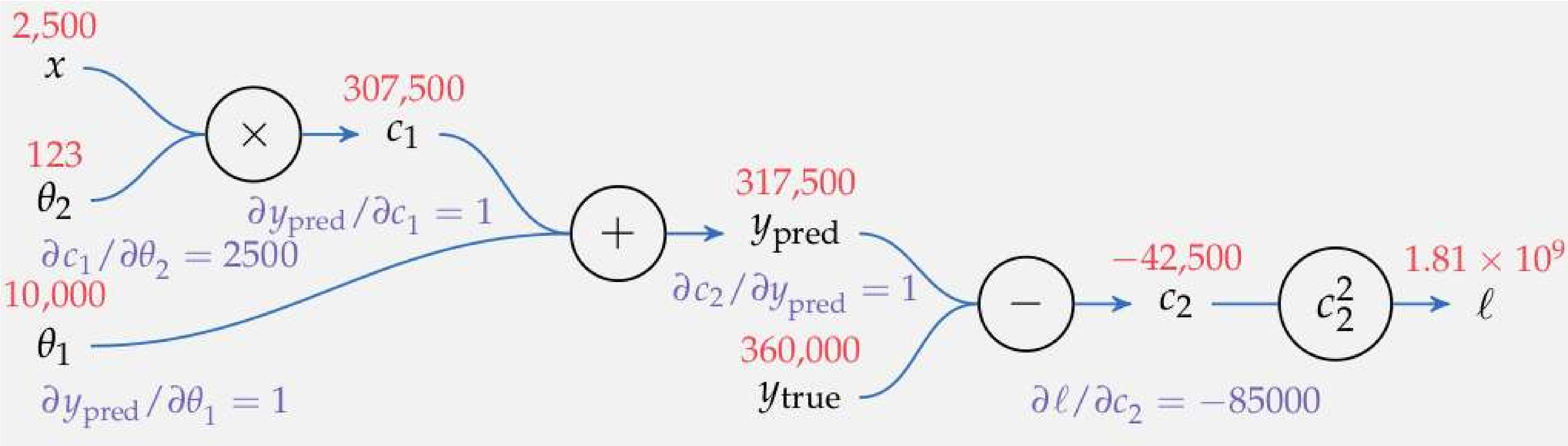
2,500
x
123
θ_2
10,000
θ_1
307,500
c_1
317,500
y_{pred}
360,000
y_{true}
−42,500
c_2
c_2^2
1.81×10^9
ℓ

Backprop



$\frac{\partial \ell}{\partial \theta_1}$ $\frac{\partial \ell}{\partial \theta_2}$

Backprop



$$\frac{\partial \ell}{\partial \theta_1} = \frac{\partial \ell}{\partial c_2} \frac{\partial c_2}{\partial y_{\text{pred}}} \frac{\partial y_{\text{pred}}}{\partial \theta_1} = -85{,}000 \cdot 1 \cdot 1 = -85{,}000$$

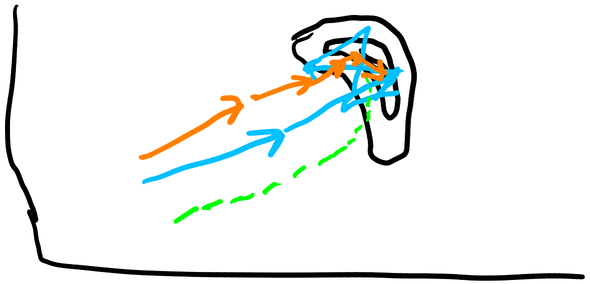
$$\frac{\partial \ell}{\partial \theta_2} = \frac{\partial \ell}{\partial c_2} \frac{\partial c_2}{\partial y_{\text{pred}}} \frac{\partial y_{\text{pred}}}{\partial c_1} \frac{\partial c_1}{\partial \theta_2} = -85{,}000 \cdot 1 \cdot 1 \cdot 2500 = -2.125 \times 10^8$$

a “fast and furious” approach to training neural networks does not work and only leads to suffering. Now, suffering is a perfectly natural part of getting a neural network to work well, but it can be mitigated by being thorough, defensive, paranoid, and obsessed with visualizations of basically every possible thing. The qualities that in my experience correlate most strongly to success in deep learning are patience and attention to detail.

- Andrej Karpathy

Adaptive Step Size: RMSProp

$$\theta = \theta + \alpha \widehat{\nabla_\theta f_\theta(x)}$$



$$\hat{s}^{(k+1)} = \gamma \hat{s}^{(k)} + (1-\gamma)(g^{(k)} \odot g^{(k)})$$

elementwise product

$$x_i^{(k+1)} = x_i^{(k)} - \frac{\alpha}{\varepsilon + \sqrt{s_i^{(k+1)}}} g_i^{(k)}$$

Adaptive Step Size: ADAM

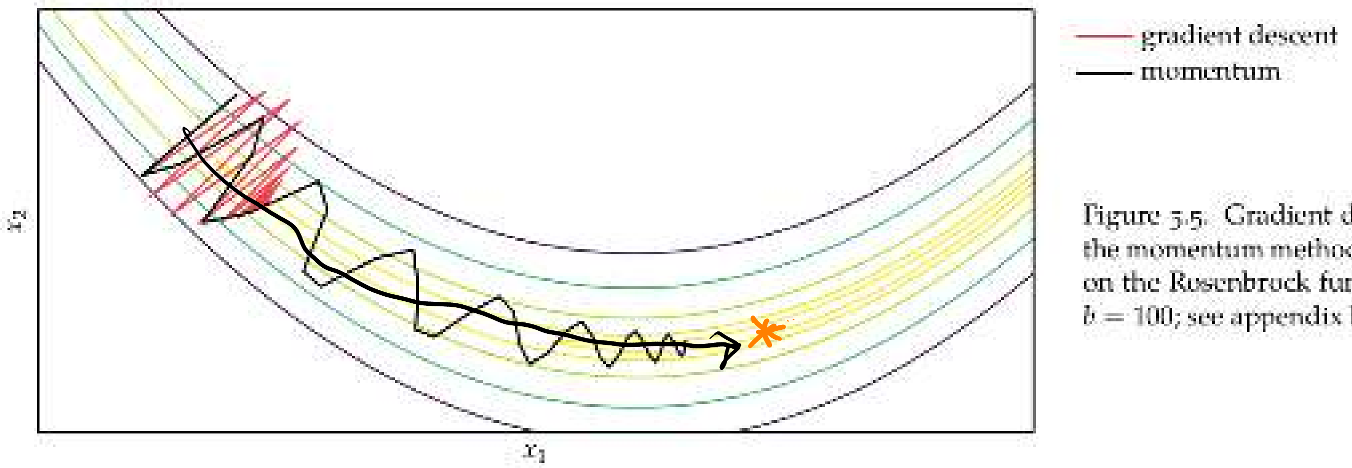


Figure 5.5. Gradient descent and the momentum method compared on the Rosenbrock function with $b = 100$; see appendix B.6.

biased decaying moment $v^{(k+1)} = \gamma_v v^{(k)} + (1-\gamma_v) g^{(k)}$

biased decaying sq. gradient $s^{(k+1)} = \gamma_s s^{(k)} + (1-\gamma_s)\left(g^{(k)} \odot g^{(k)}\right)$

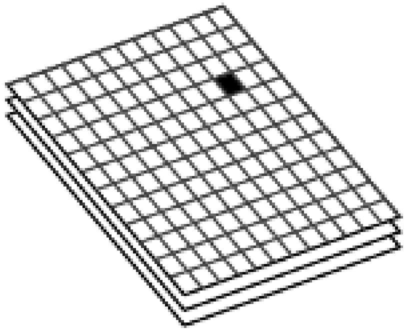
corrected d. m. $\hat{v}^{(k+1)} = v^{(k+1)} / (1-\gamma_v^k)$

corrected sq. gradient $\hat{s} = s / (1-\gamma_s^k)$

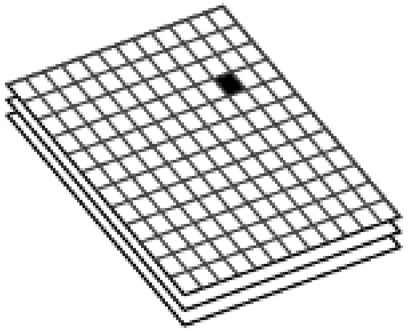
$$x^{(k+1)} = x^{(k)} + \alpha \hat{v}^{(k+1)} \Big/ \left(\varepsilon + \sqrt{\hat{s}^{(k+1)}}\right)$$

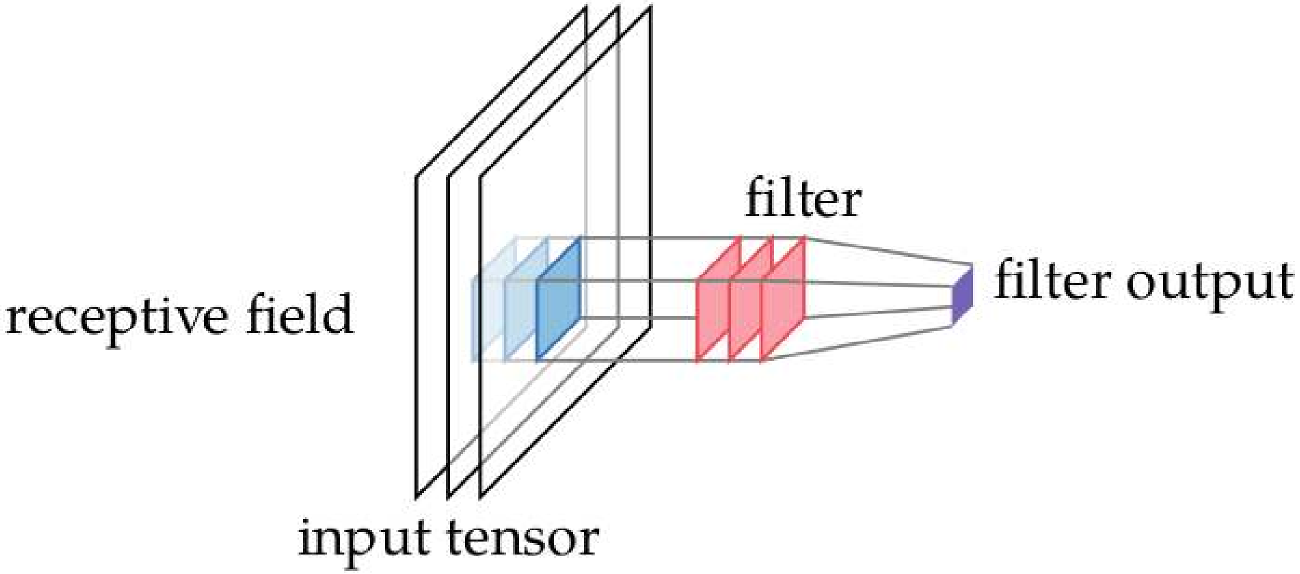
On Your Radar: ConvNets

On Your Radar: ConvNets

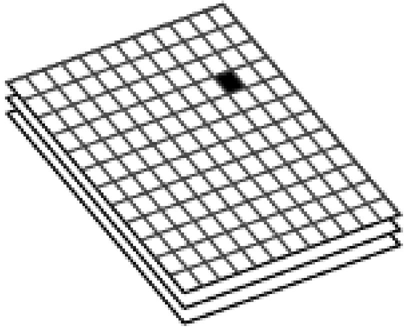


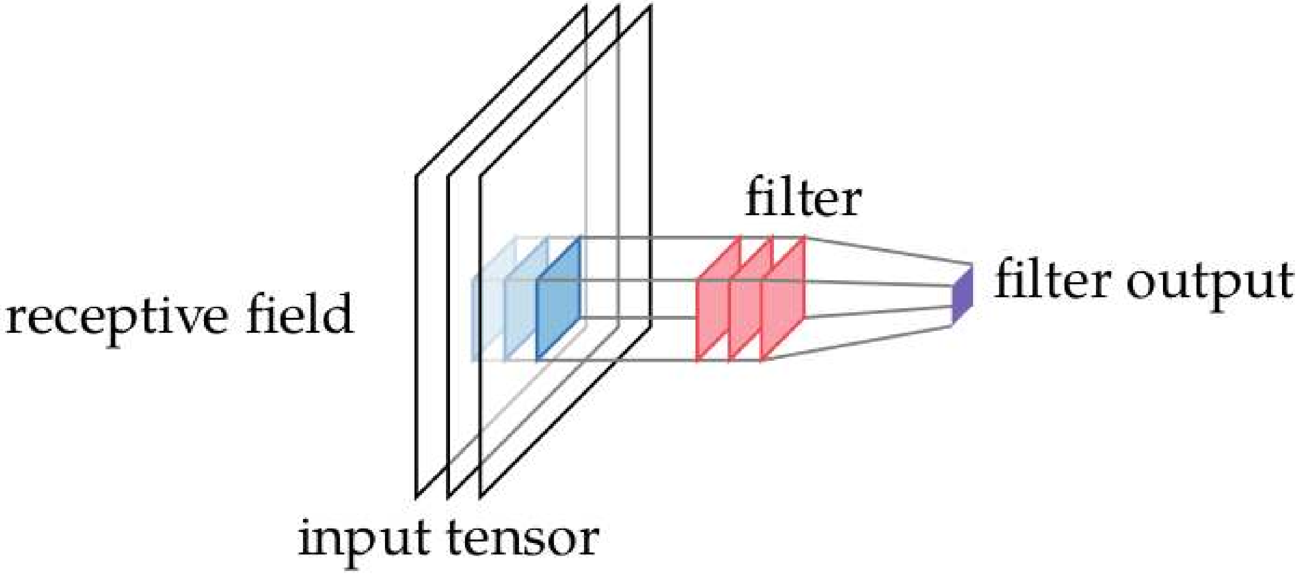
On Your Radar: ConvNets

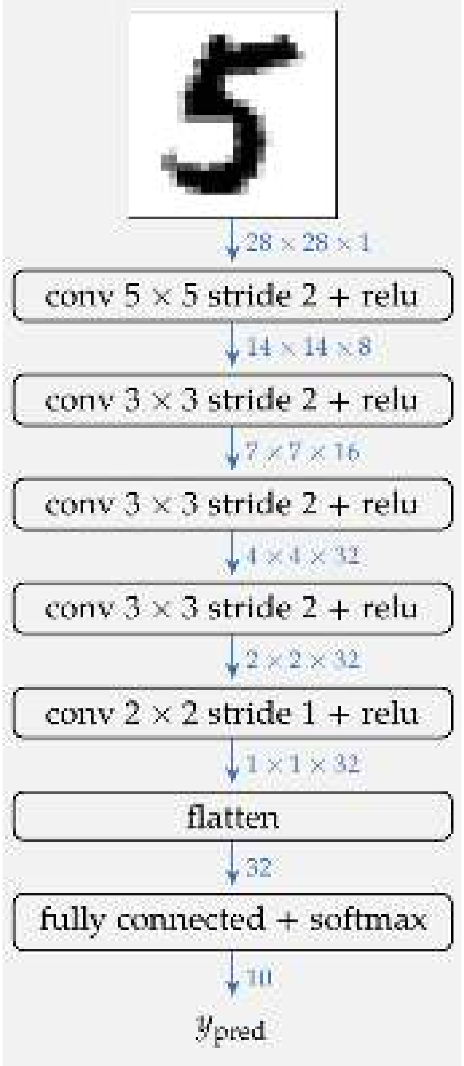




On Your Radar: ConvNets







On Your Radar: Regularization

On Your Radar: Regularization

$$\arg\min_{\theta} \sum_{(x,y)\in\mathbf{D}} \ell(f_\theta(x), y) - \beta\|\theta\|^2$$

On Your Radar: Regularization

$$\arg\min_{\theta} \sum_{(x,y)\in\mathbf{D}} \ell(f_{\theta}(x), y) - \beta\|\theta\|^2$$

e.g. Batch norm, dropout