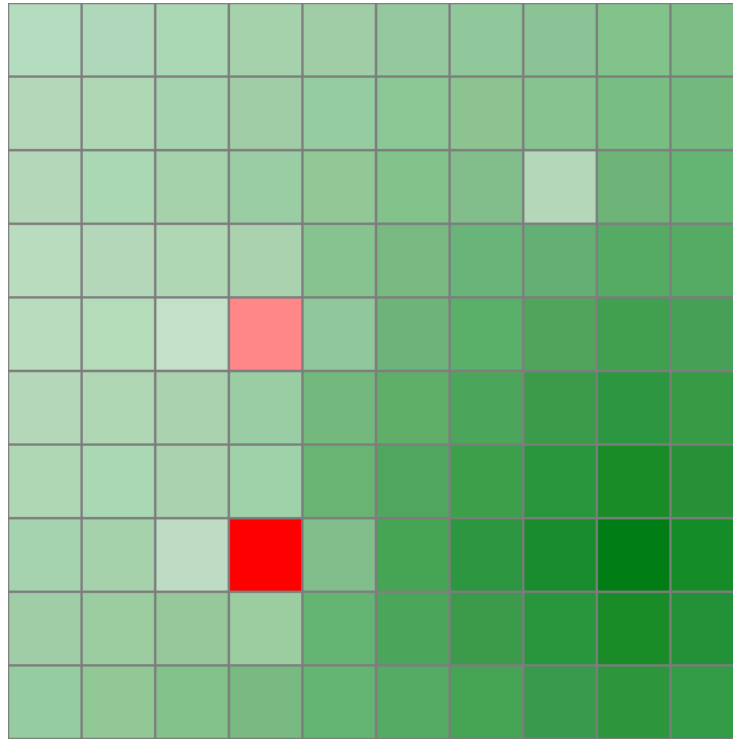


Neural Network Function Approximation




Do we really need to keep track of $U(s)$ for every U separately?

Function Approximation

Function Approximation

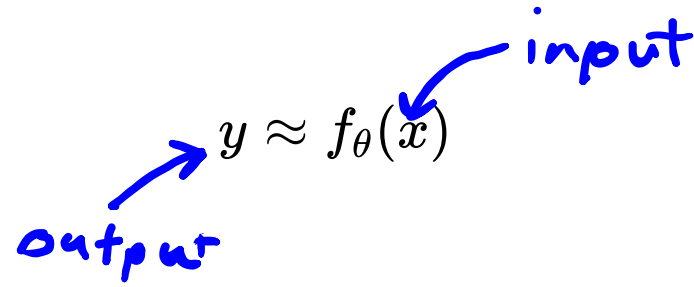
$$y \approx f_{\theta}(x)$$

Function Approximation

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


input

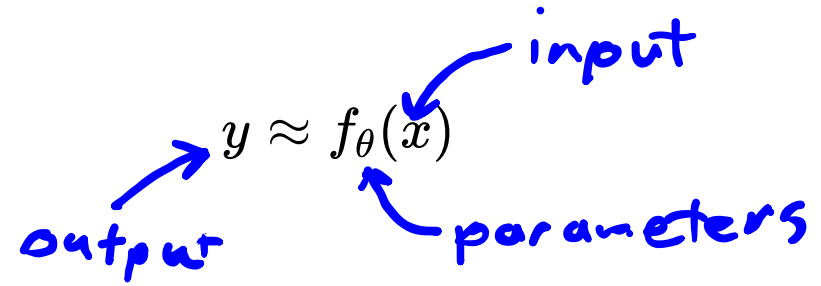
Function Approximation



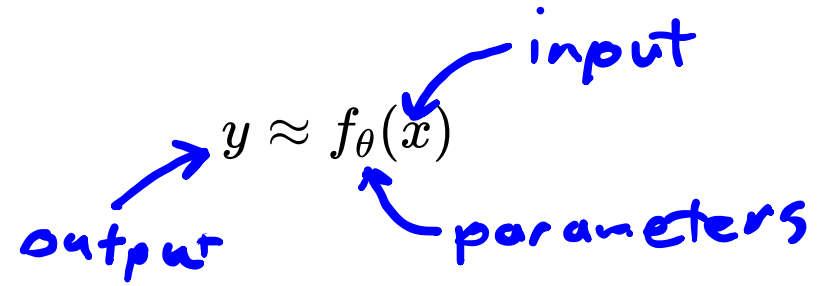
A diagram illustrating the function approximation equation $y \approx f_{\theta}(x)$. The equation is centered, with a handwritten blue arrow pointing from the word "output" to the variable y on the left, and another handwritten blue arrow pointing from the word "input" to the variable x inside the function $f_{\theta}(x)$ on the right.

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

Function Approximation



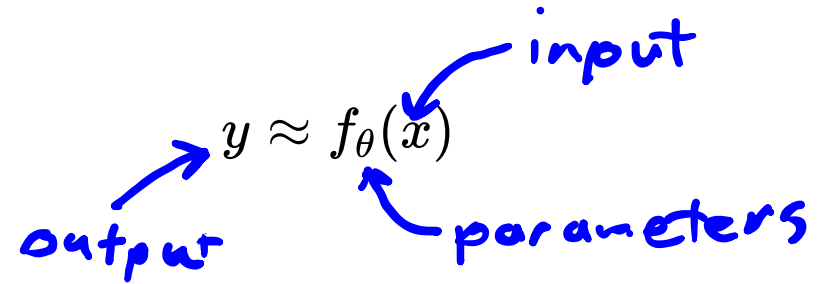
Function Approximation



Example: Linear Function Approximation:

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

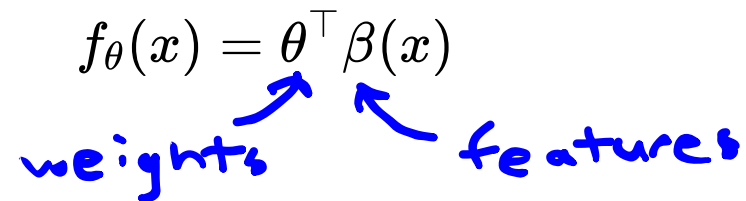
Function Approximation



A diagram showing the equation $y \approx f_{\theta}(x)$. Three blue arrows point to the components: one from the word "output" to y , one from the word "input" to x , and one from the word "parameters" to θ .

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

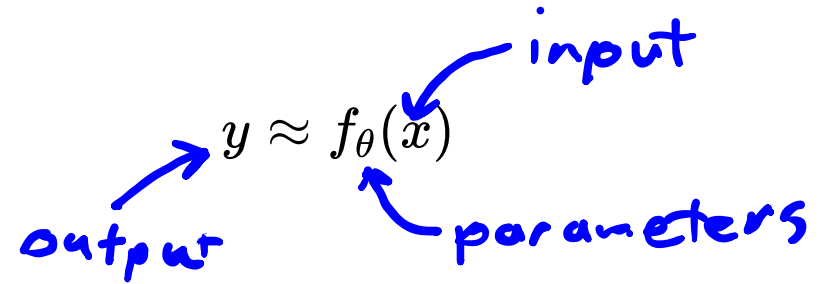
Example: Linear Function Approximation:



A diagram showing the equation $f_{\theta}(x) = \theta^{\top} \beta(x)$. Two blue arrows point to the components: one from the word "weights" to θ , and one from the word "features" to $\beta(x)$.

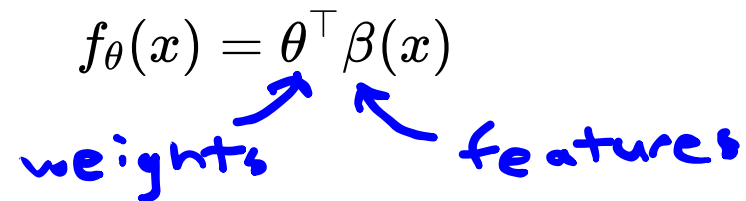
$$f_{\theta}(x) = \theta^{\top} \beta(x)$$

Function Approximation



A diagram showing the general function approximation equation $y \approx f_{\theta}(x)$. Three blue arrows point to the components: 'input' points to x , 'parameters' points to θ , and 'output' points to y .

Example: Linear Function Approximation:



A diagram showing the linear function approximation equation $f_{\theta}(x) = \theta^{\top} \beta(x)$. Two blue arrows point to the components: 'weights' points to θ and 'features' points to $\beta(x)$.

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

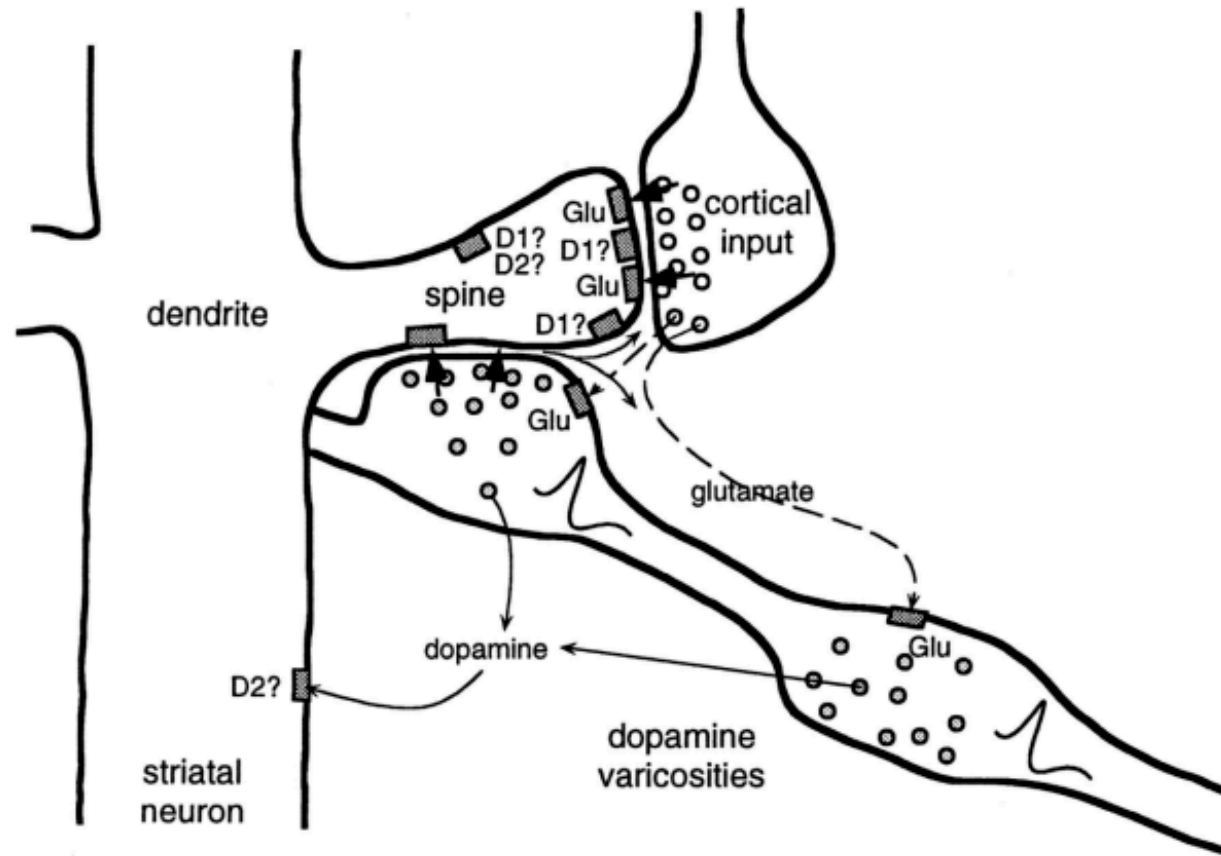
Neural Network

Neural Network

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

Neural Network

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

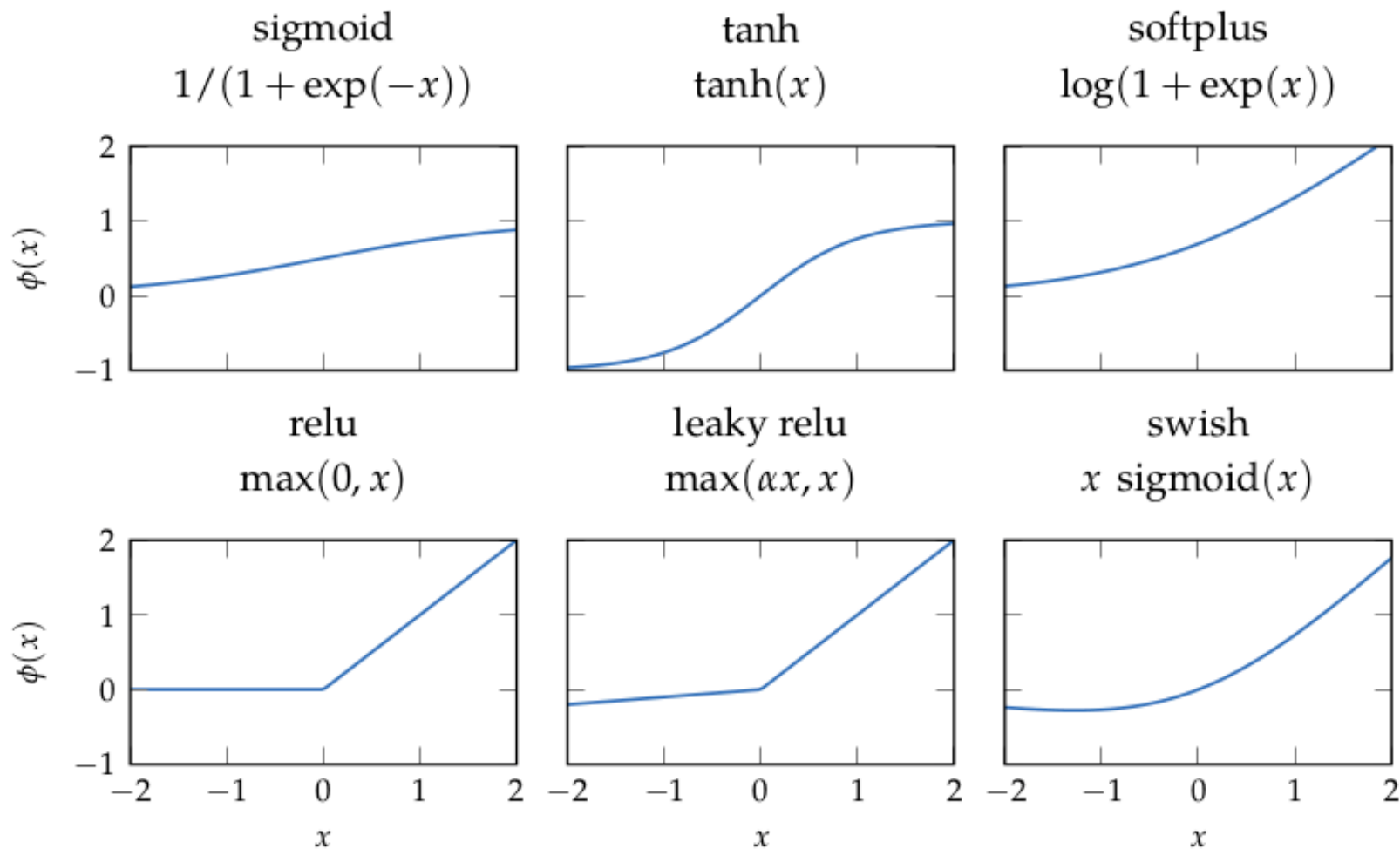


Neural Network

Neural Network

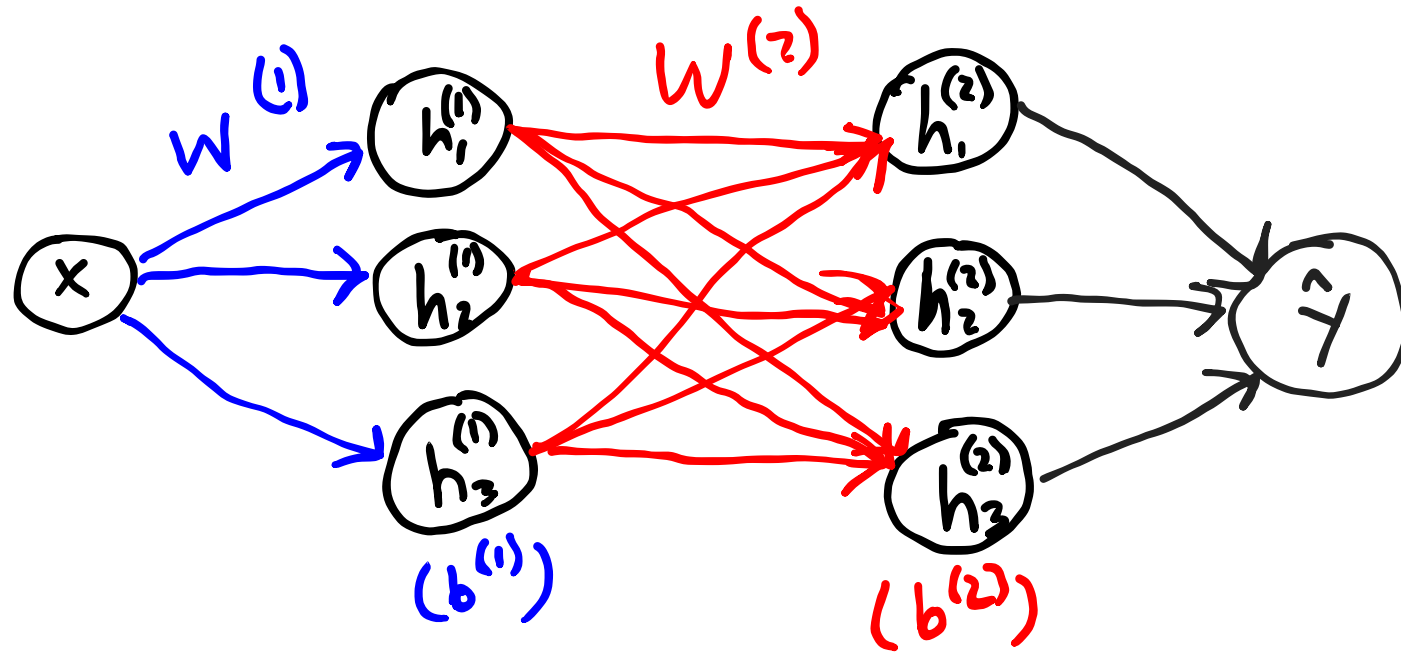
$$h(x) = \sigma(Wx + b)$$

Nonlinearities

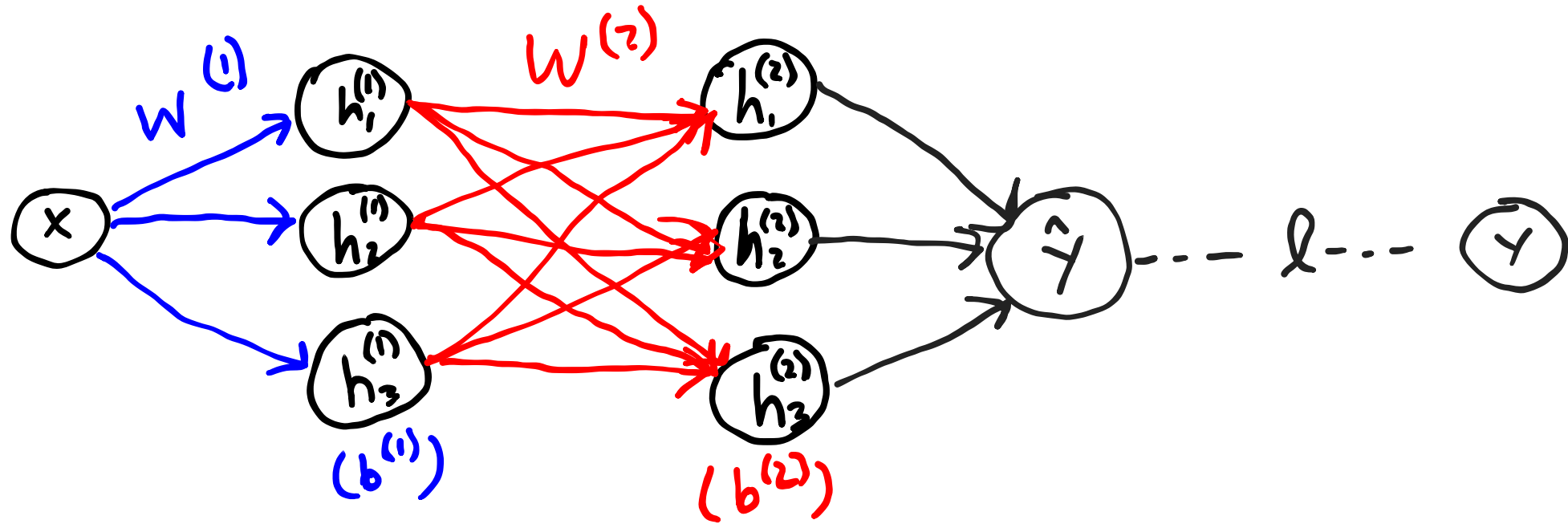


Training

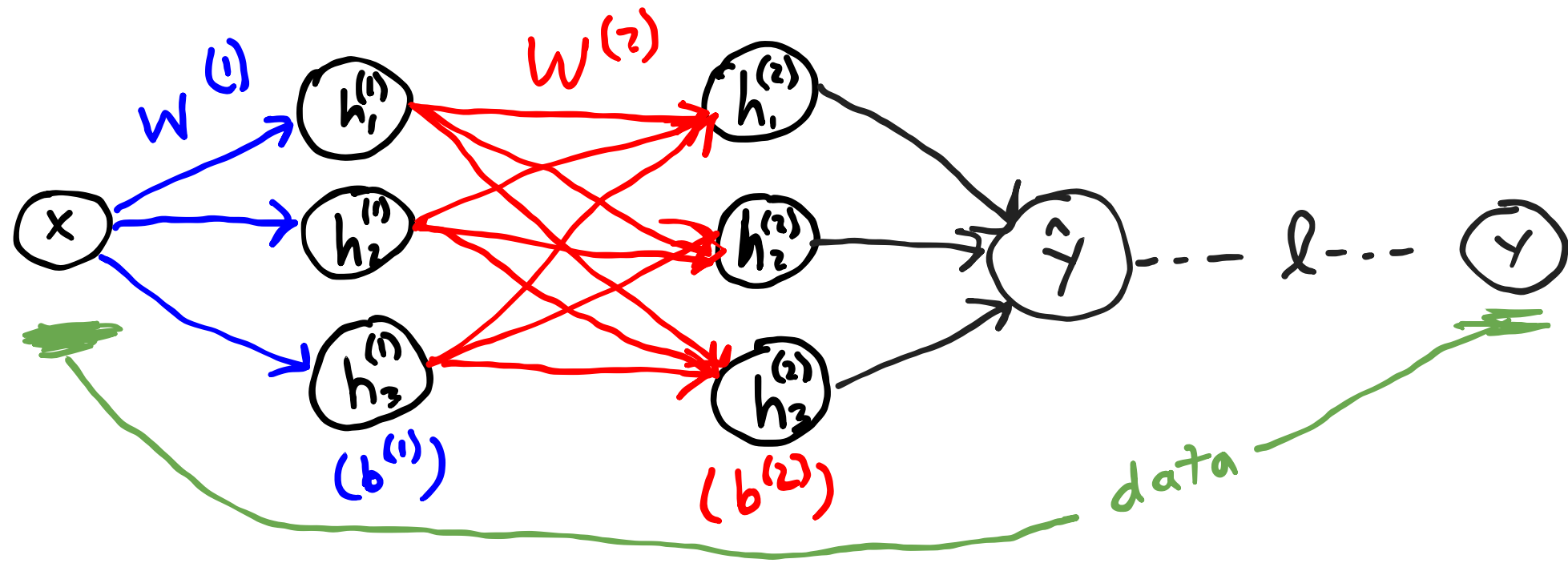
Training



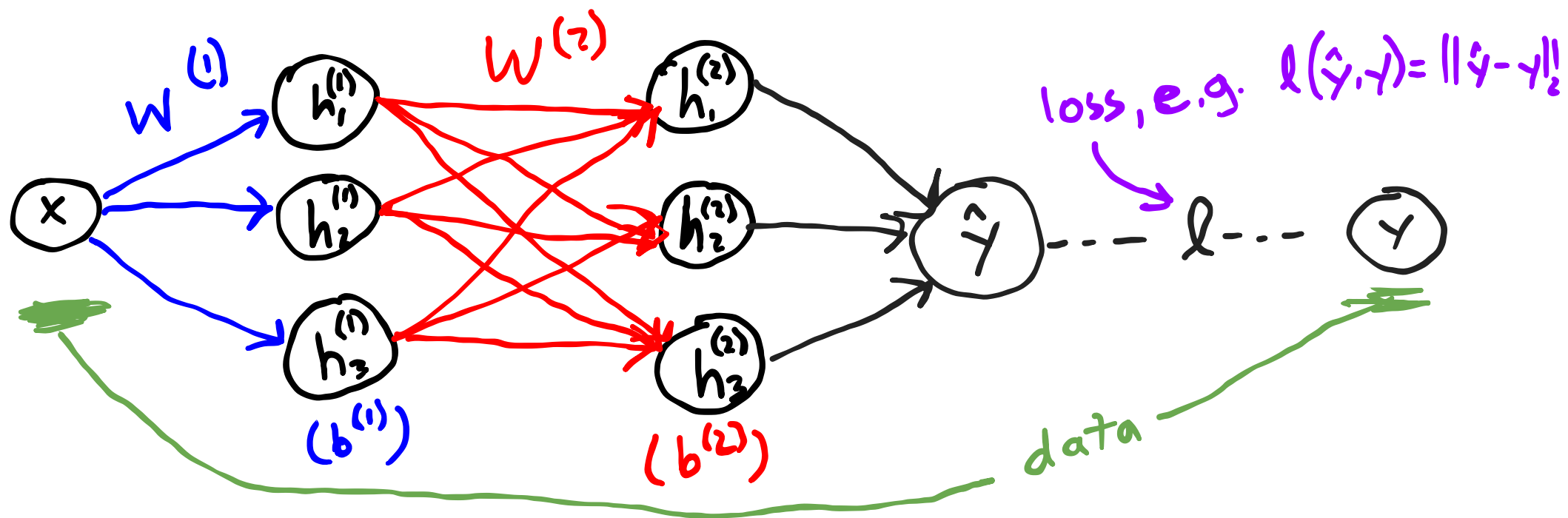
Training



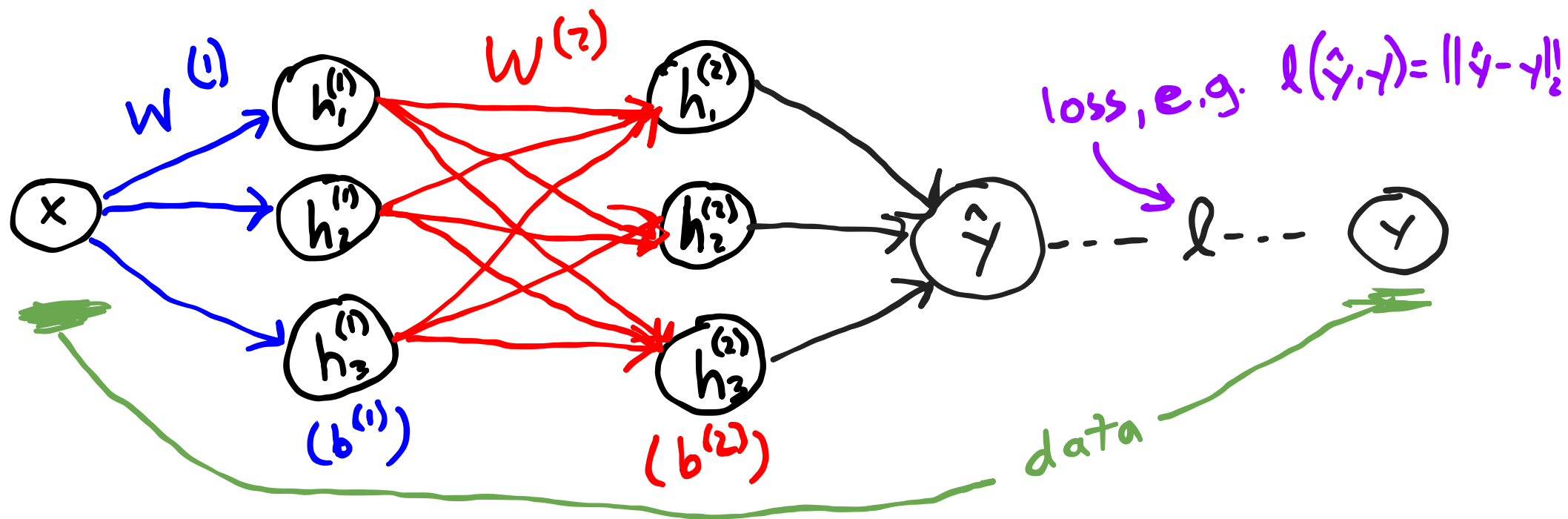
Training



Training

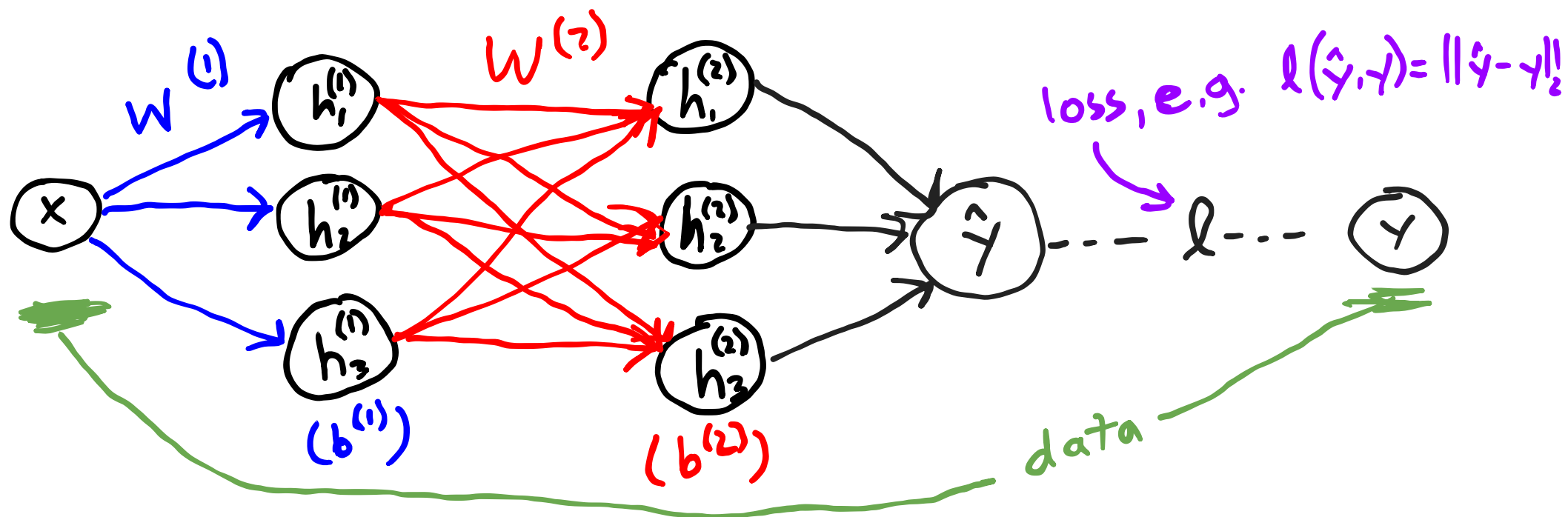


Training



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

Training



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

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

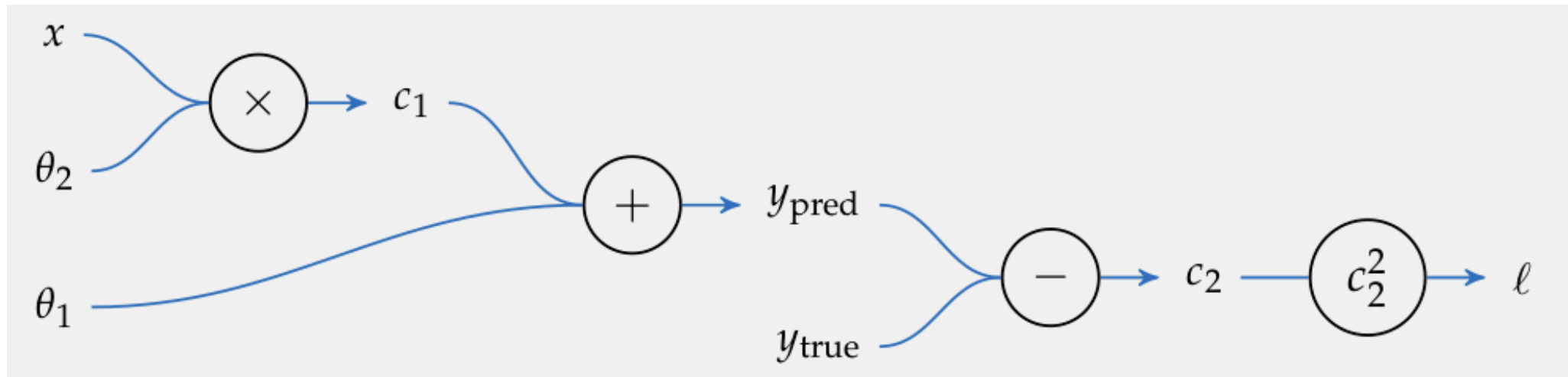
Chain Rule

Backprop

$$l(x, y_{\text{true}}) = (\theta_2 x + \theta_1 - y_{\text{true}})^2$$

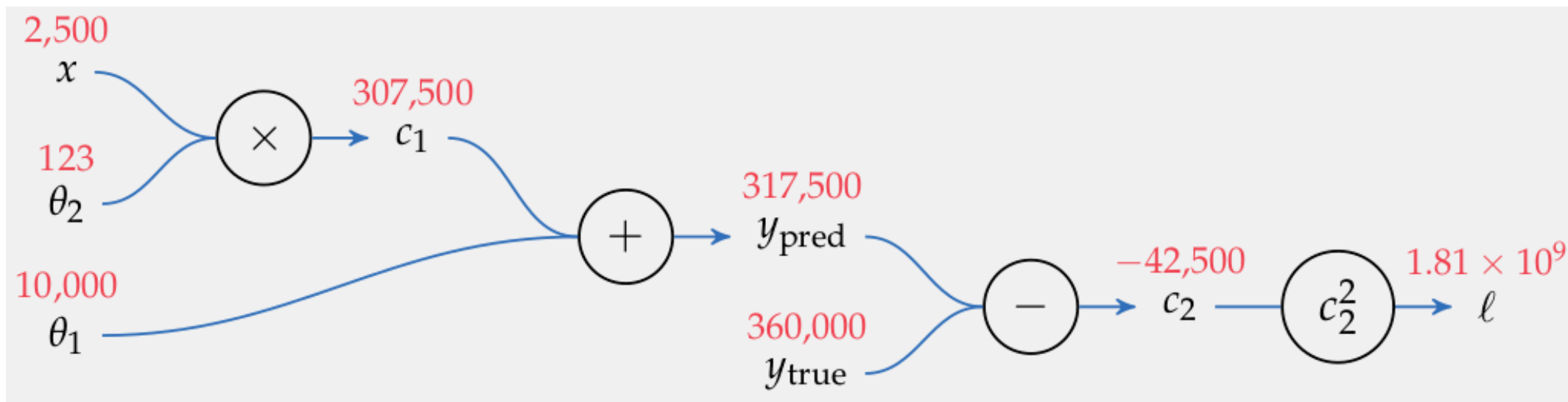
Backprop

$$l(x, y_{\text{true}}) = (\theta_2 x + \theta_1 - y_{\text{true}})^2$$



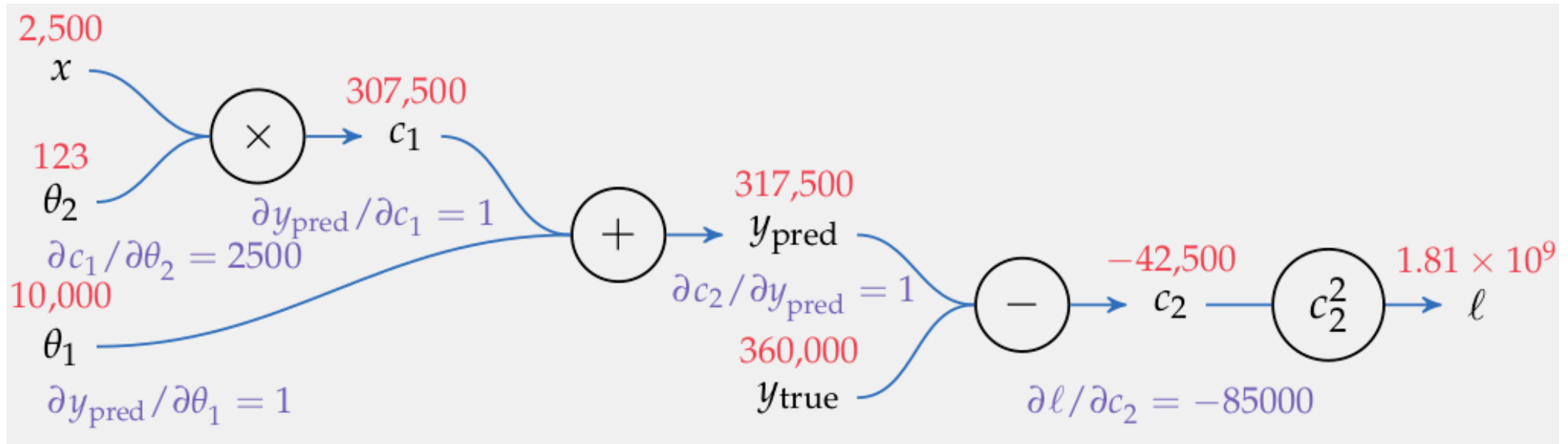
Backprop

$$l(x, y_{\text{true}}) = (\theta_2 x + \theta_1 - y_{\text{true}})^2$$



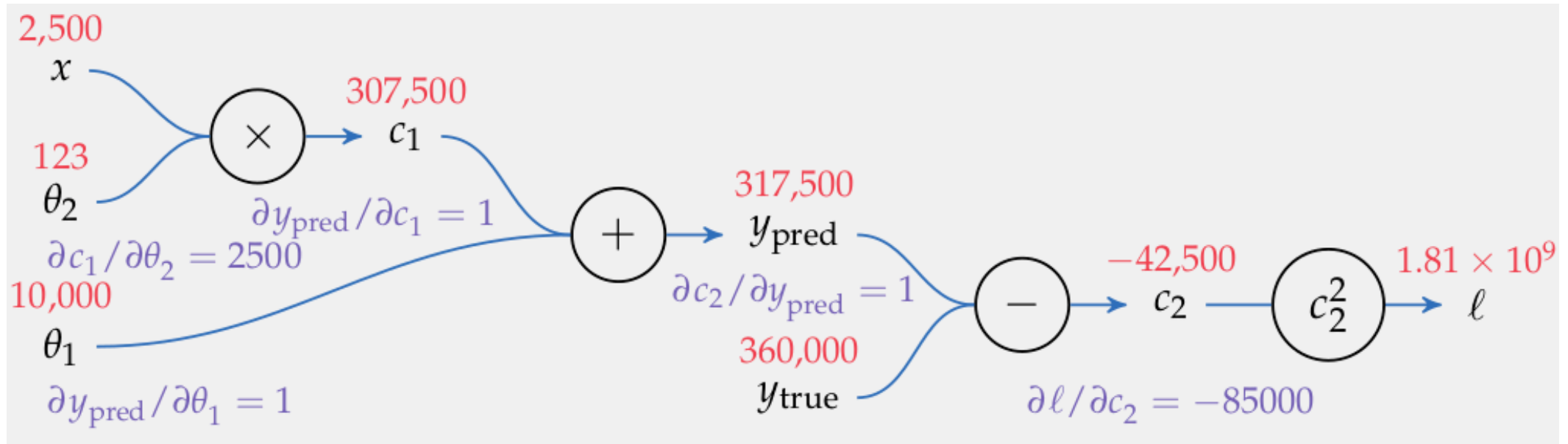
Backprop

$$l(x, y_{\text{true}}) = (\theta_2 x + \theta_1 - y_{\text{true}})^2$$



Backprop

$$l(x, y_{\text{true}}) = (\theta_2 x + \theta_1 - y_{\text{true}})^2$$



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

$$\frac{\partial \ell}{\partial \theta_2} = \frac{\partial \ell}{\partial c_2} \frac{\partial c_2}{\partial y_{\text{pred}}} \frac{\partial y_{\text{pred}}}{\partial c_1} \frac{\partial c_1}{\partial \theta_2} = -85,000 \cdot 1 \cdot 1 \cdot 2,500 = -2.125 \times 10^8$$

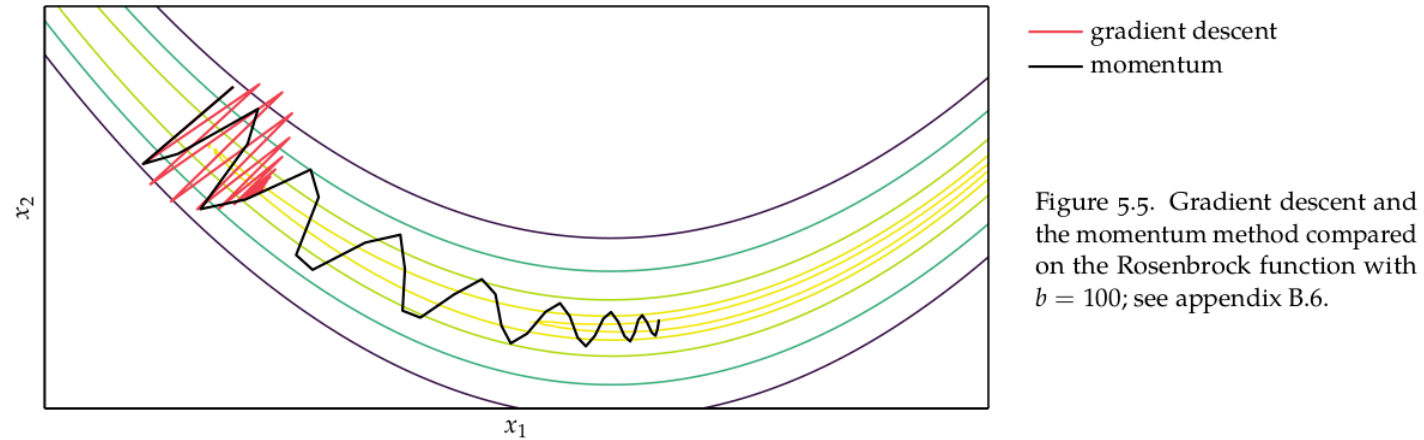
a “fast and furious” approach to training neural networks does not work and only leads to suffering. Now, suffering is a perfectly natural part of getting a neural network to work well, but it can be mitigated by being thorough, defensive, paranoid, and obsessed with visualizations of basically every 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

Adaptive Step Size: ADAM

(Adaptive Moment Estimation)



Adaptive Step Size: ADAM

(Adaptive Moment Estimation)

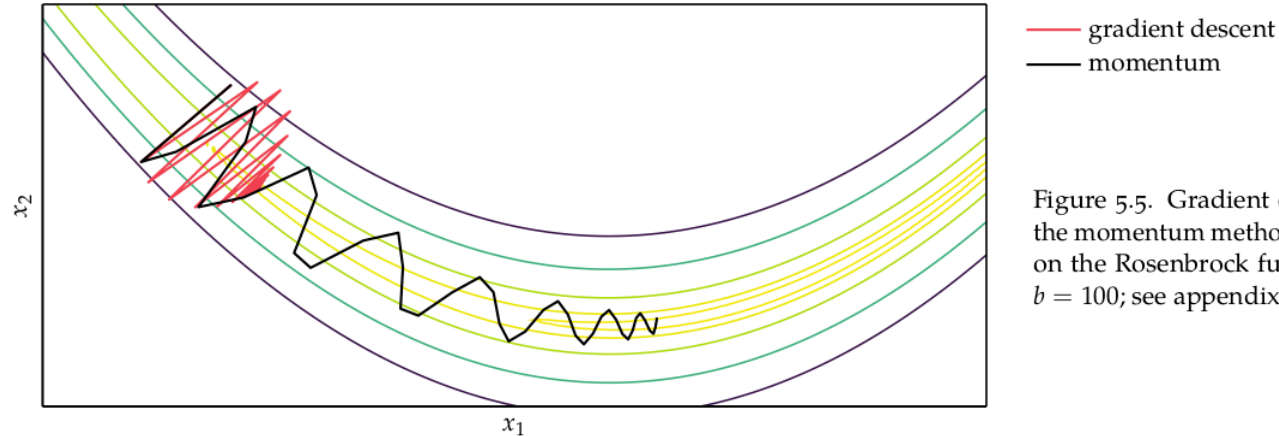


Figure 5.5. Gradient descent and the momentum method compared on the Rosenbrock function with $b = 100$; see appendix B.6.

$$\text{biased decaying momentum: } \mathbf{v}^{(k+1)} = \gamma_v \mathbf{v}^{(k)} + (1 - \gamma_v) \mathbf{g}^{(k)} \quad (5.29)$$

$$\text{biased decaying sq. gradient: } \mathbf{s}^{(k+1)} = \gamma_s \mathbf{s}^{(k)} + (1 - \gamma_s) (\mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}) \quad (5.30)$$

$$\text{corrected decaying momentum: } \hat{\mathbf{v}}^{(k+1)} = \mathbf{v}^{(k+1)} / (1 - \gamma_v^k) \quad (5.31)$$

$$\text{corrected decaying sq. gradient: } \hat{\mathbf{s}}^{(k+1)} = \mathbf{s}^{(k+1)} / (1 - \gamma_s^k) \quad (5.32)$$

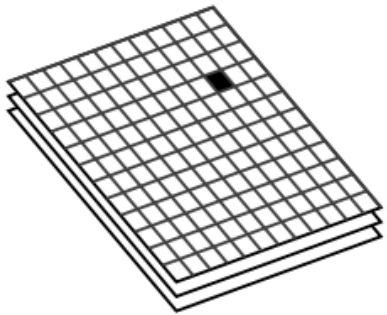
$$\text{next iterate: } \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \hat{\mathbf{v}}^{(k+1)} / \left(\epsilon + \sqrt{\hat{\mathbf{s}}^{(k+1)}} \right) \quad (5.33)$$

¹² According to the original paper, good default settings are $\alpha = 0.001$, $\gamma_v = 0.9$, $\gamma_s = 0.999$, and $\epsilon = 1 \times 10^{-8}$.

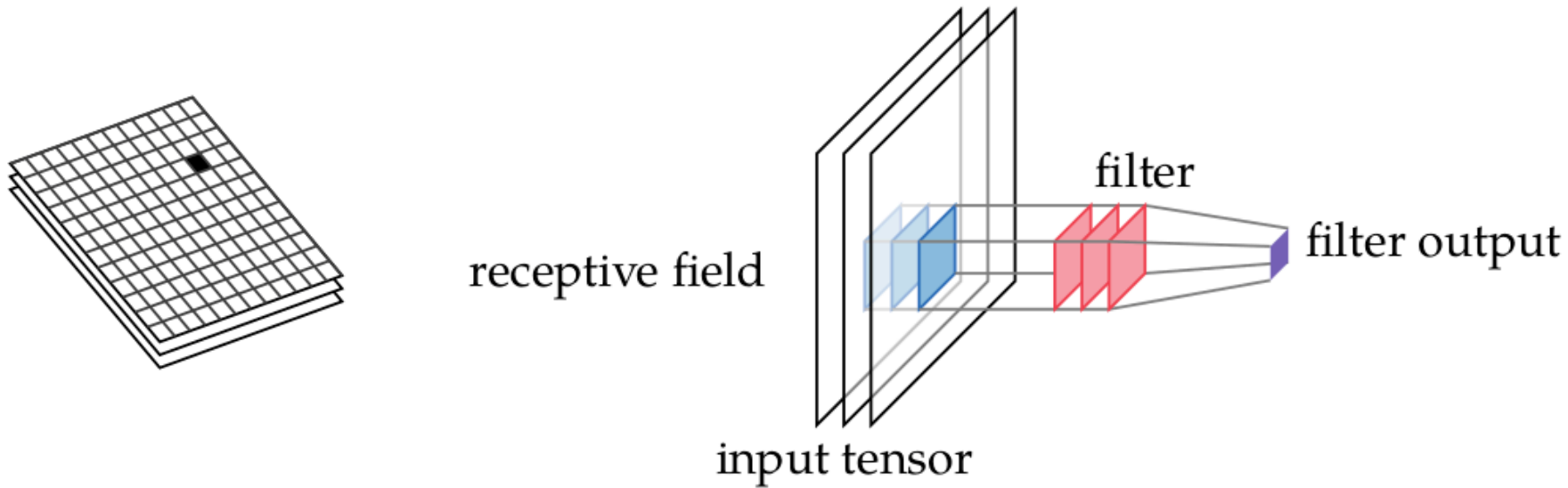
\odot means elementwise multiplication.

On Your Radar: ConvNets

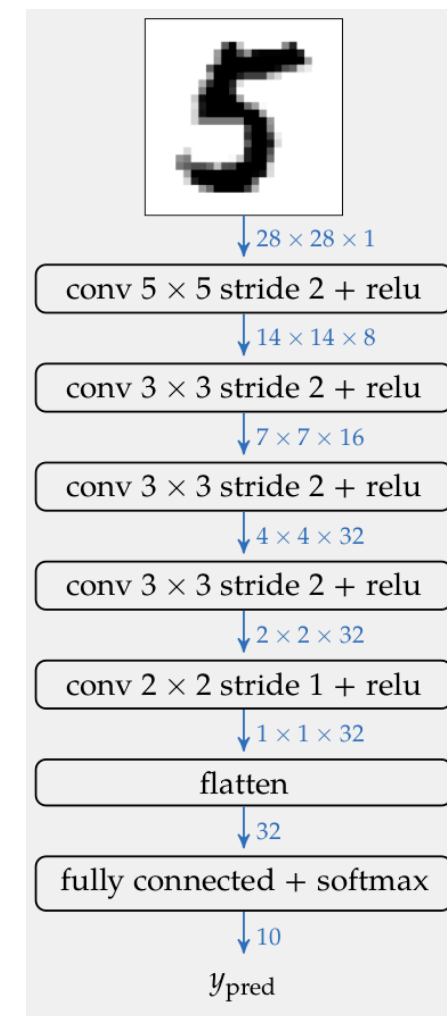
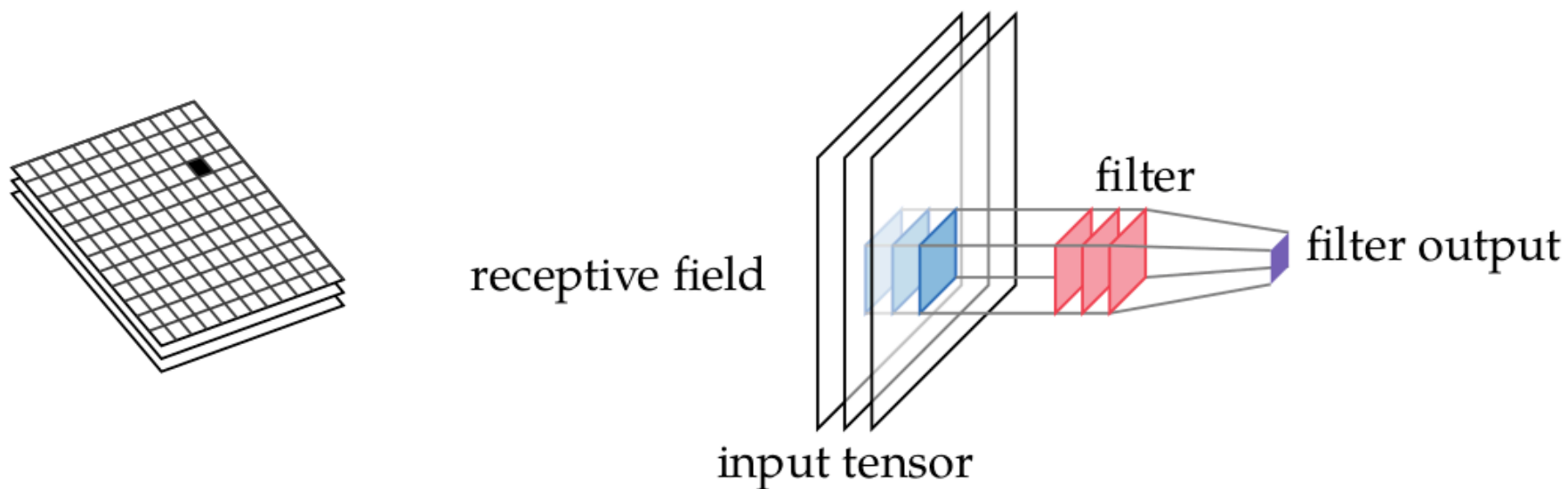
On Your Radar: ConvNets



On Your Radar: ConvNets



On Your Radar: ConvNets



On Your Radar: Regularization

On Your Radar: Regularization

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

On Your Radar: Regularization

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

e.g. Batch norm, layer norm, dropout

On Your Radar: Skip Connections (Resnets)

Resources

OpenAI Spinning up