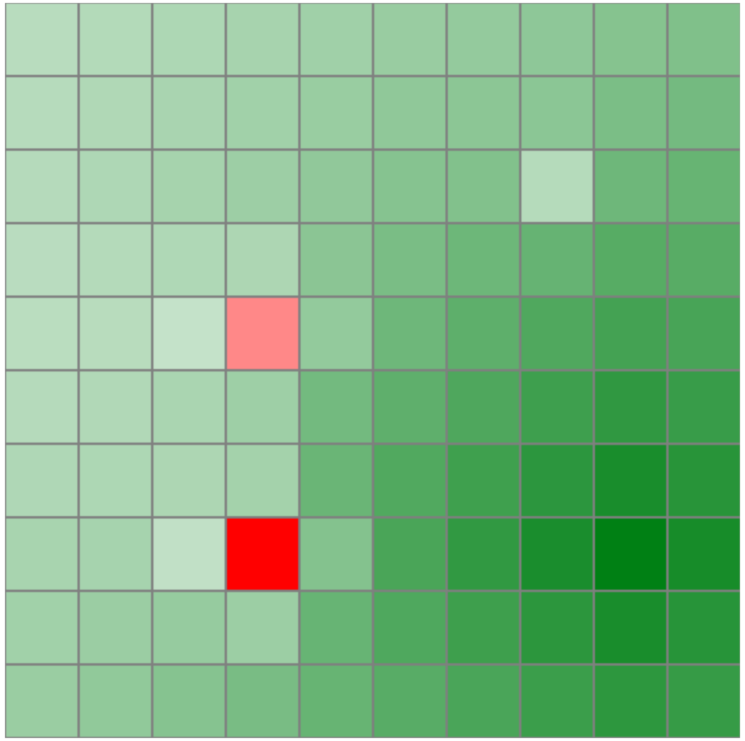
# Neural Network Function Approximation



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

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

Example: Linear Function Approximation:

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

# Function Approximation

input

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

output parameters

Example: Linear Function Approximation:

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

weights features

# Function Approximation

input

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

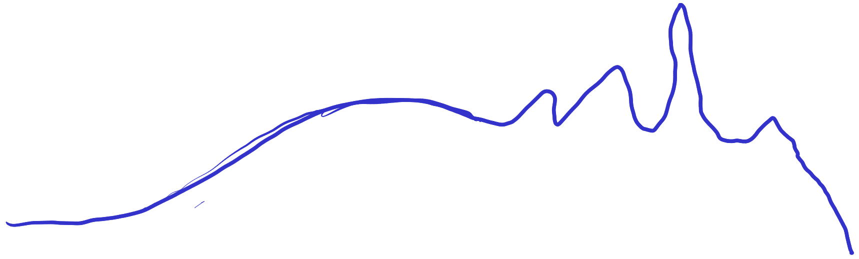
output parameters

Example: Linear Function Approximation:

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

weights features

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



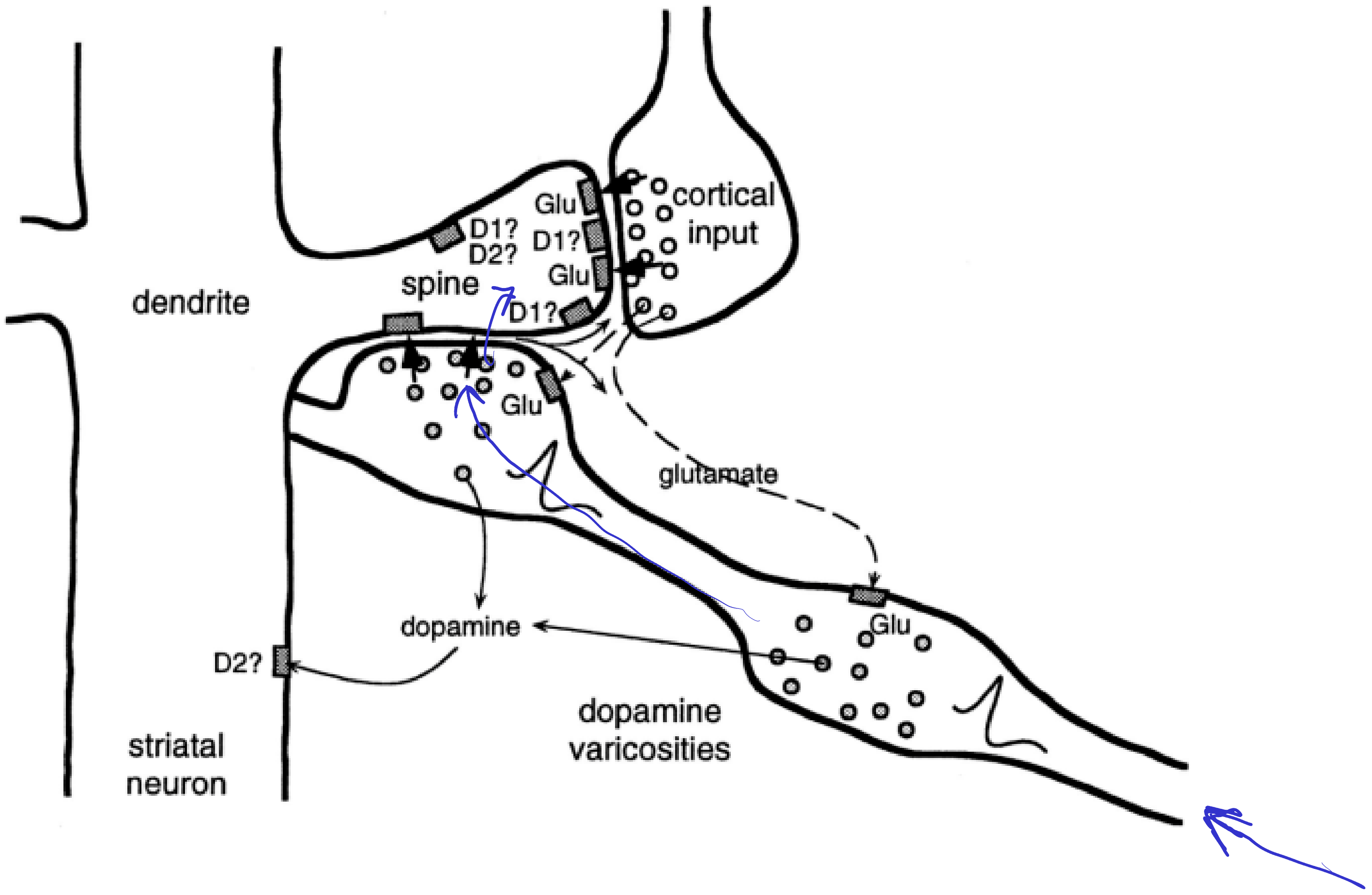
# Neural Network

# Neural Network

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

# Neural Network

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



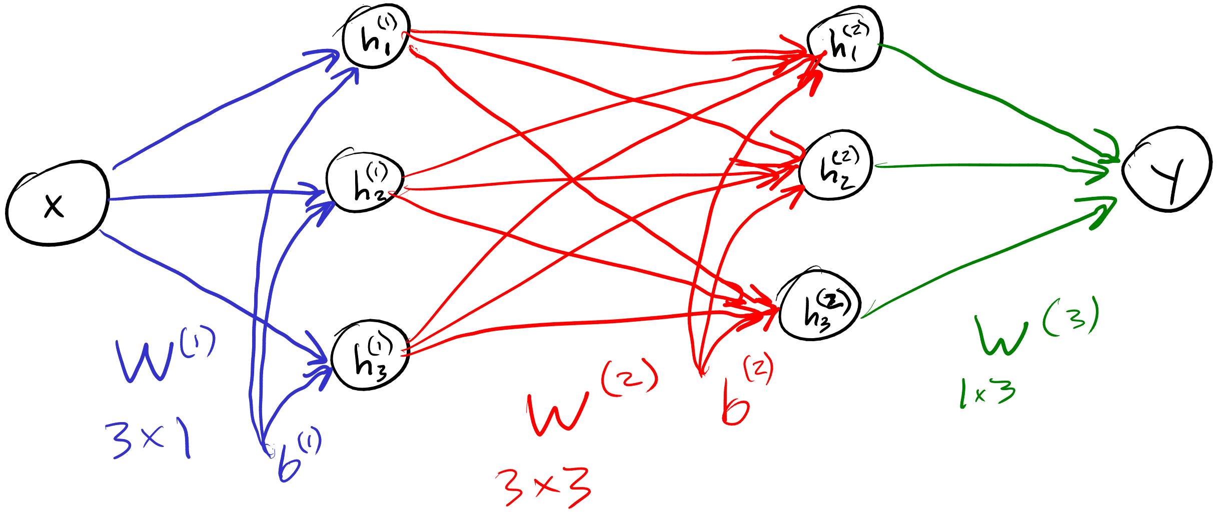
# Neural Network

# Neural Network

if $\sigma(x)=x$

$y = W^{*}x + b^{*}$

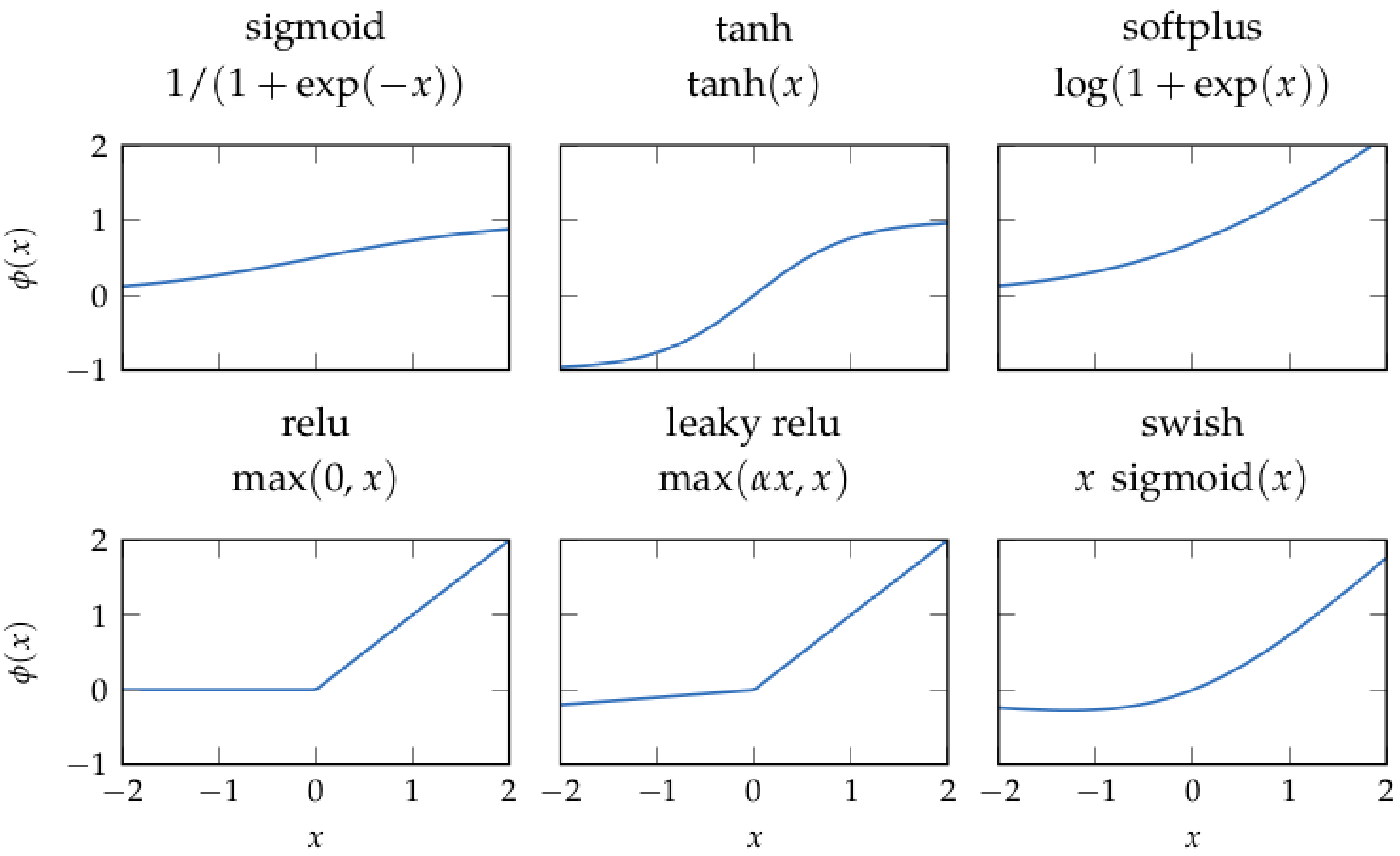
$$h(x) = \sigma(Wx + b)$$




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

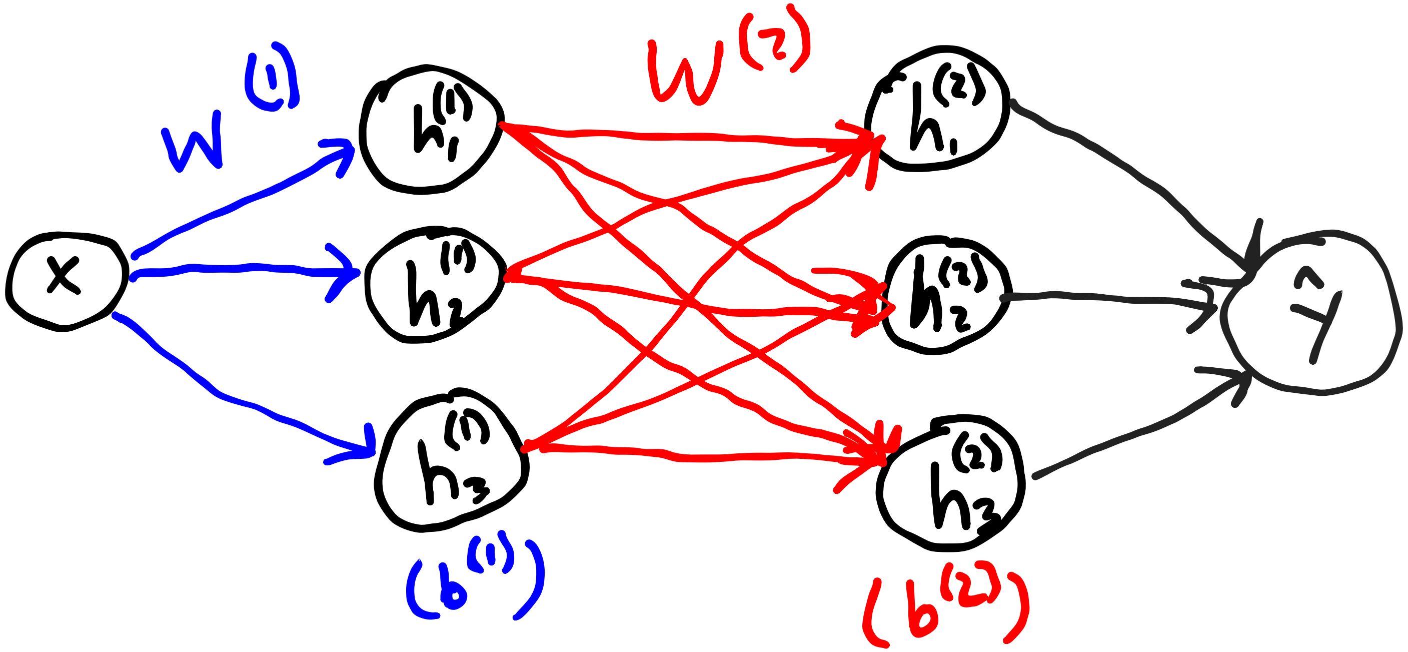
$$\theta = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, W^{(3)})$$

# Nonlinearities

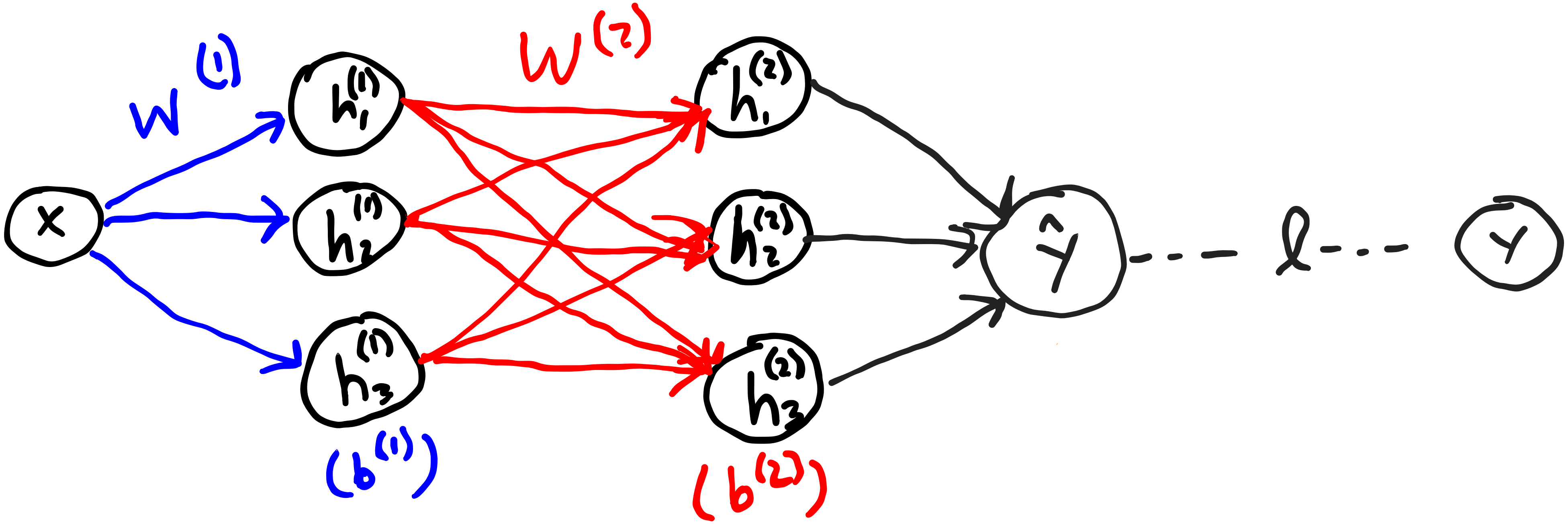


# Training

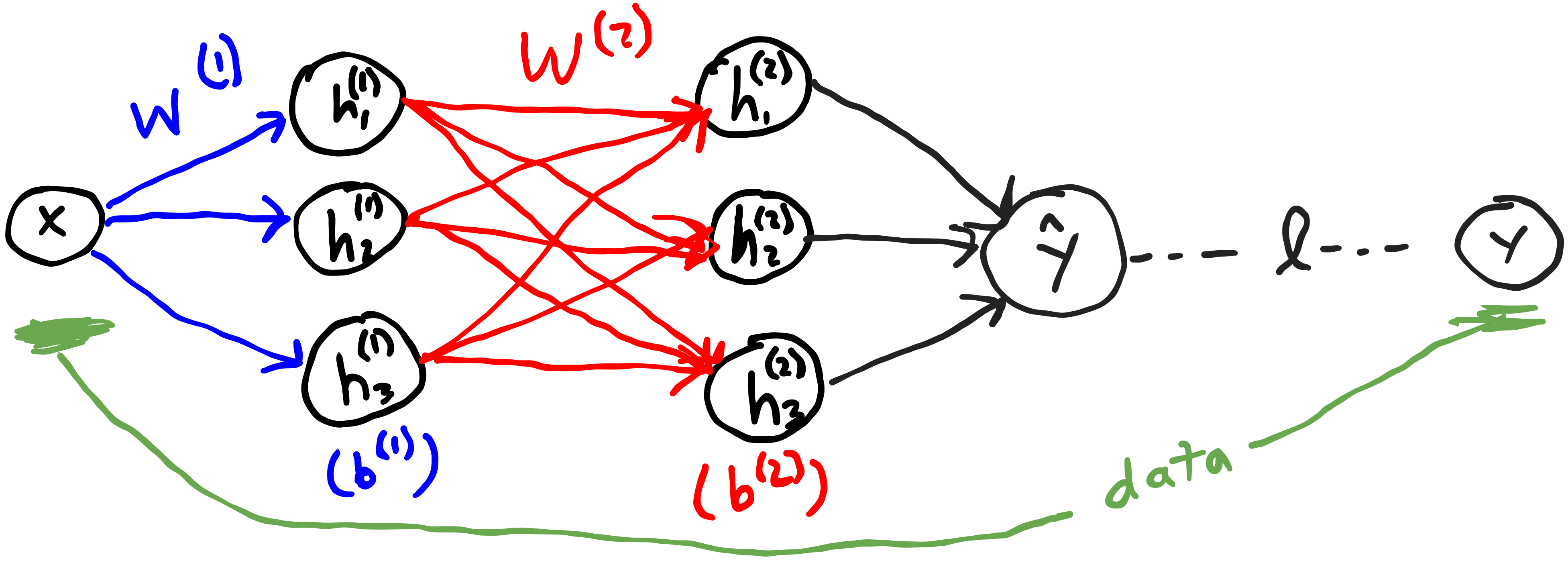
# Training



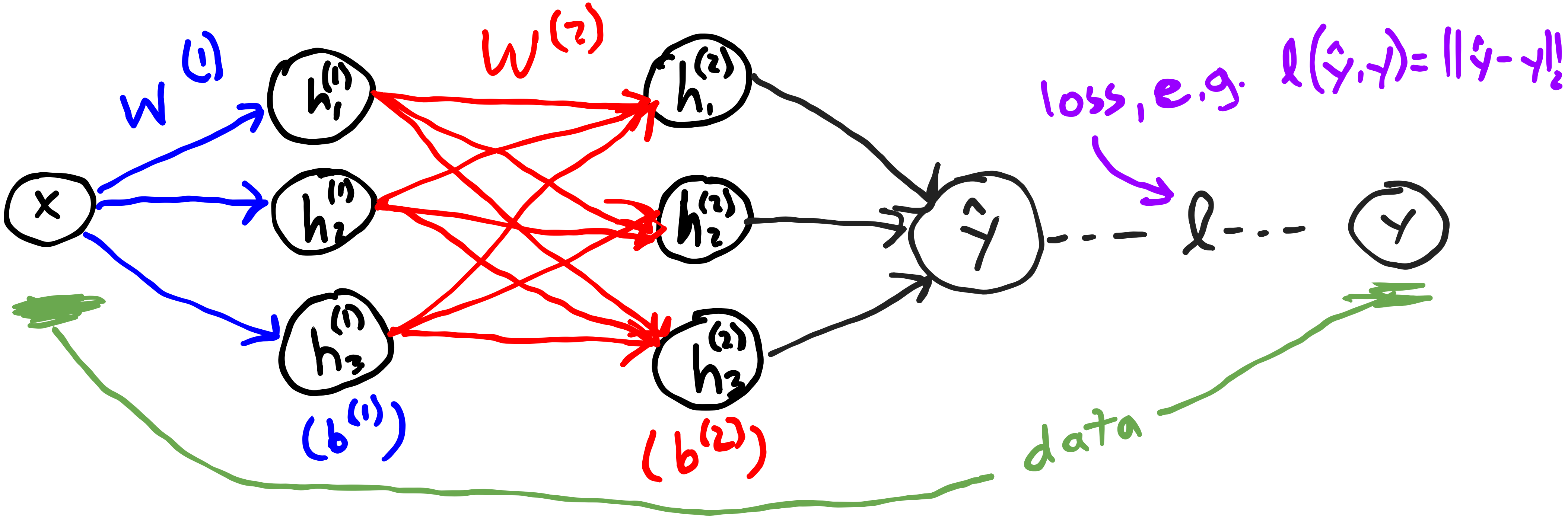
# Training



# Training

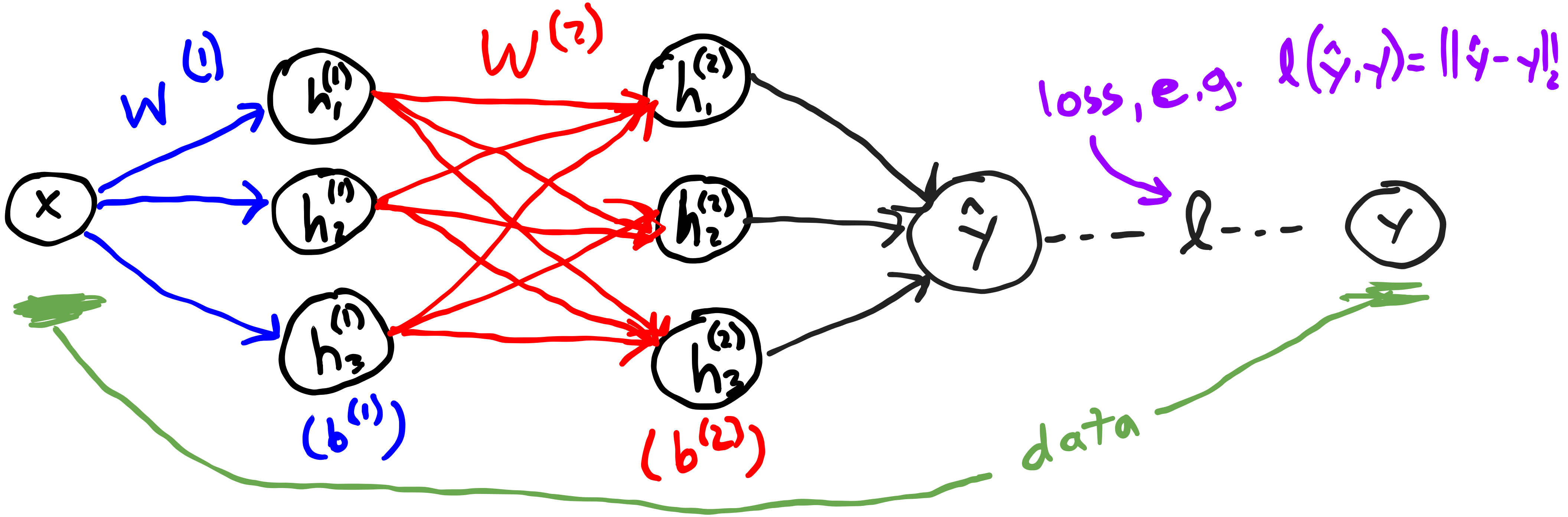


# Training



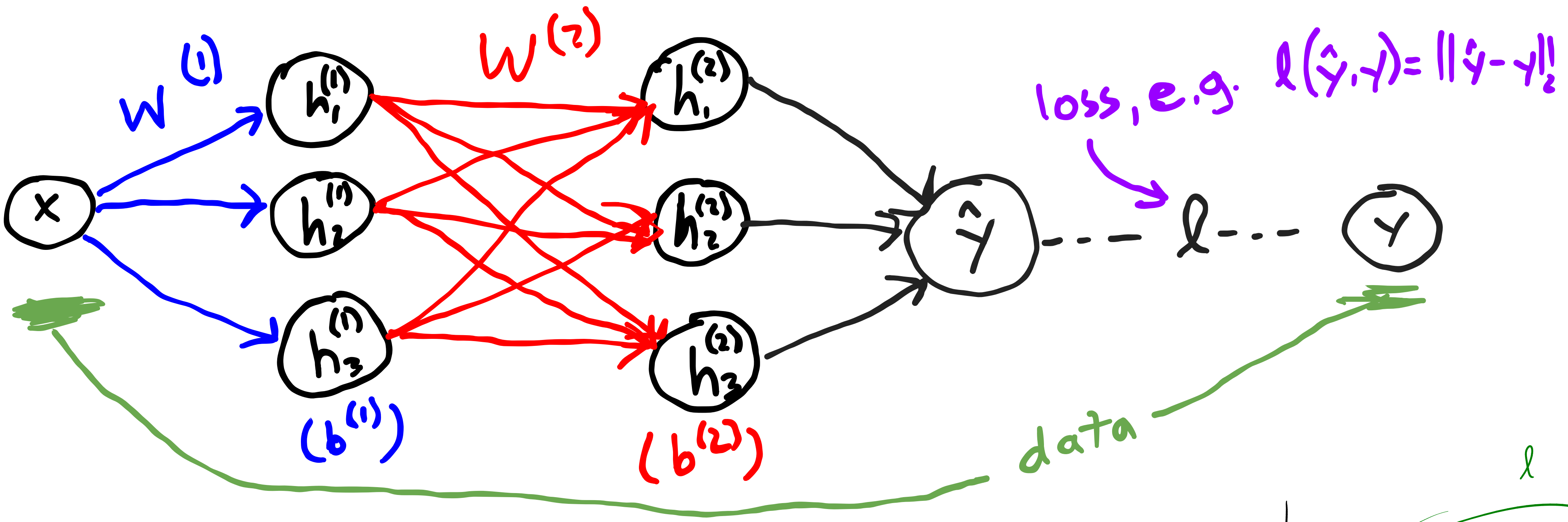
$W^{(1)}$
$W^{(2)}$
$x$
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_1^{(2)}$
$h_2^{(2)}$
$h_3^{(2)}$
$(b^{(1)})$
$(b^{(2)})$
$\hat{y}$
$\ell$
$y$
loss, e.g. $\ell(\hat{y}, y) = \|\hat{y} - y\|_2^2$
data

# Training

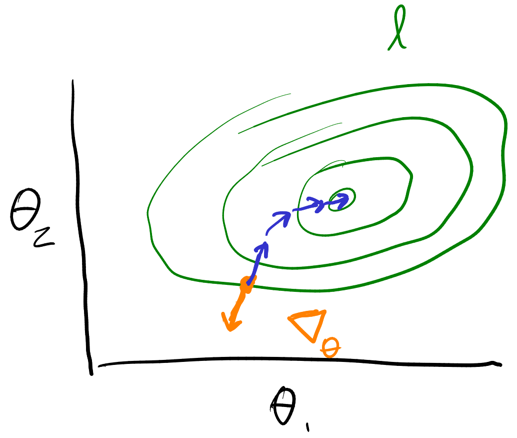



$$\theta^* = \arg\min_{\theta} \sum_{(x,y)\in\mathcal{D}} l(f_\theta(x), y)$$

# Training




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




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

learning rate

$\theta = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, W^{(3)})$

$$\nabla_\theta \ell = \begin{bmatrix} \frac{\partial \ell}{\partial \theta_1} \\ \vdots \\ \frac{\partial \ell}{\partial \theta_n} \end{bmatrix}$$

# Chain Rule

$f(g(h(x)))$

$$\left.\frac{\partial f(g(h(x)))}{\partial x}\right|_{x_0} = \left.\frac{\partial f(g(h))}{\partial h}\right|_{h_0} \left.\frac{\partial h(x)}{\partial x}\right|_{x_0} = \left.\frac{\partial f(g)}{\partial g}\right|_{g_0} \left.\frac{\partial g(h)}{\partial h}\right|_{h_0} \left.\frac{\partial h(x)}{\partial x}\right|_{x_0}$$

$$\ell(x,y) = (f_\theta(x) - y)^2$$

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

$$\left.\frac{\partial \ell}{\partial W^{(2)}}\right|_0 = \left.\frac{\partial \ell}{\partial f}\right|_0 \left.\frac{\partial f}{\partial W^{(2)}}\right|_0 = 2(f_\theta(x_0) - y_0)\, \underline{\sigma(W^{(1)}x_0 + b^{(1)})}$$

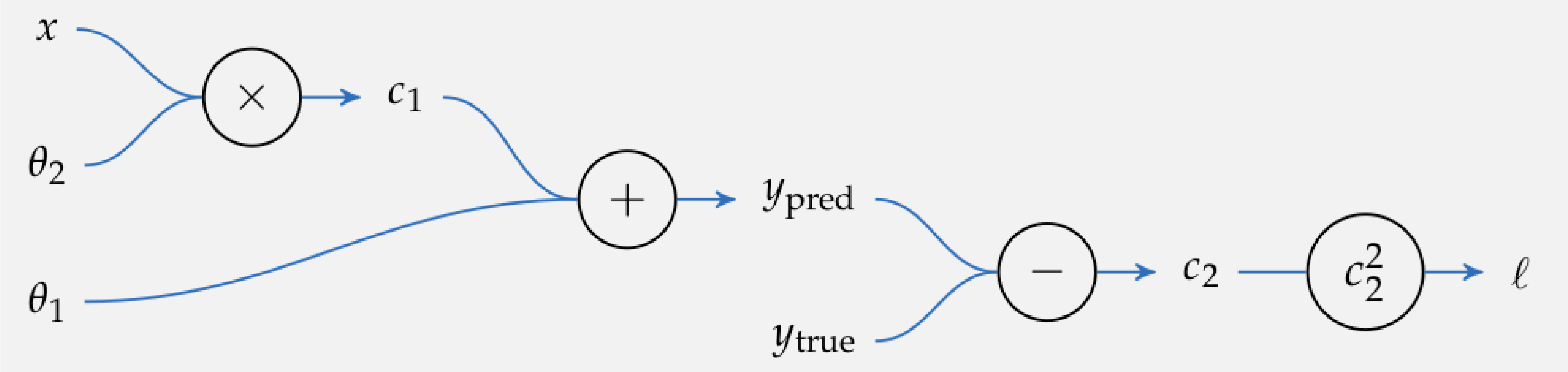
output ~~of~~ $h^{(1)}(x_0)$

# Backprop

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

# Backprop

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

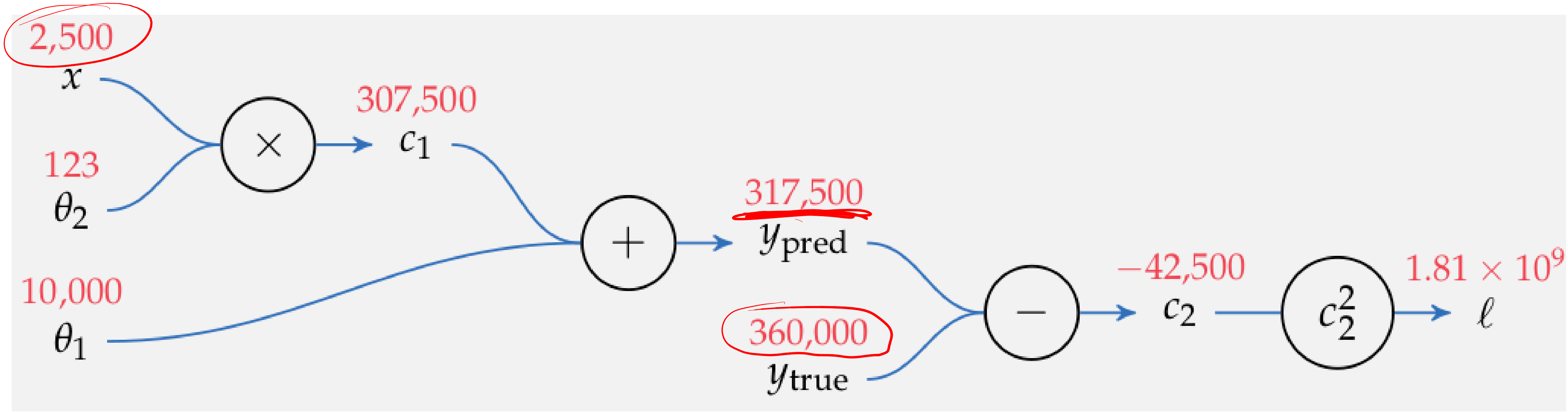



forward pass $f_\theta(x)$, $l$

backward pass $\frac{\partial l}{\partial \theta_i}$

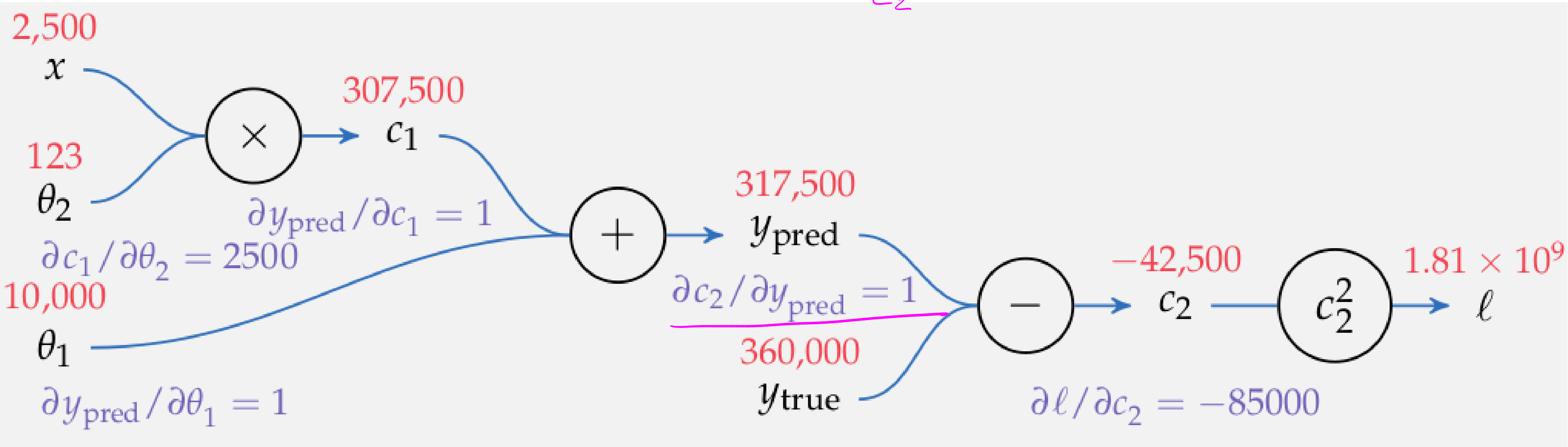
# Backprop

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



# Backprop

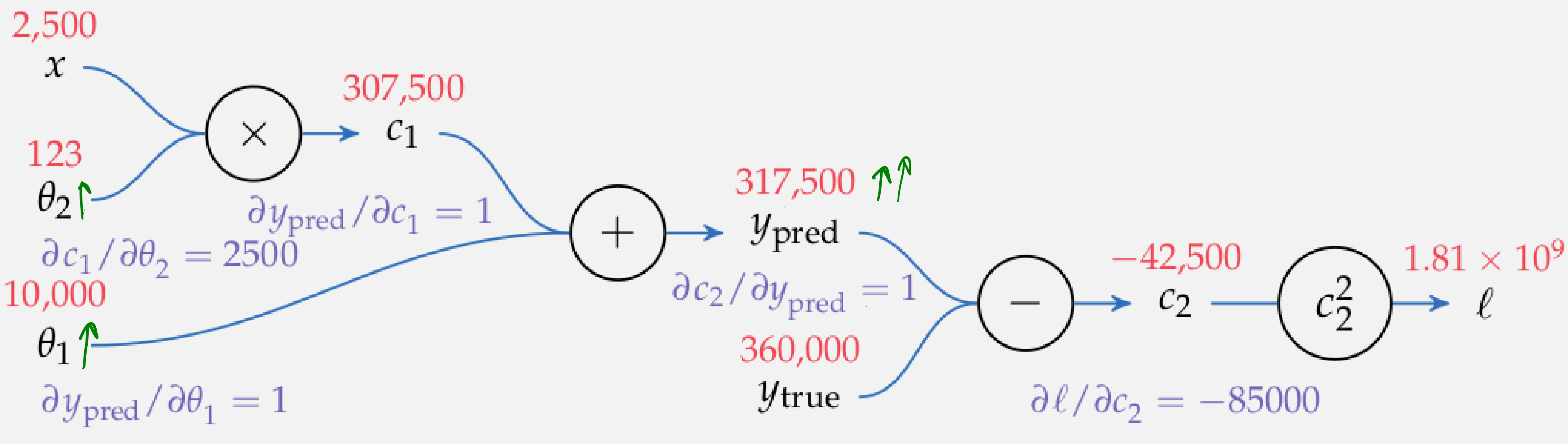
$$l(x, y_{\text{true}}) = \underbrace{(\theta_2 x + \theta_1 - y_{\text{true}})}_{c_2}{}^2$$




$$\frac{\partial \ell}{\partial c_2} = 2 \cdot c2 =$$

# Backprop

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




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

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

$$\nabla_\theta \ell = \begin{bmatrix} -85{,}000 \\ -2.125 \times 10^8 \end{bmatrix}$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta \ell$$

```julia
function train(x_data, y_data;
    learning_rate=1e-3, n_epochs=1_000, save_every=50, minibatch_size=1
    )

    model = Chain(
        Dense(1=>32, tanh),
        Dense(32=>32, tanh),
        Dense(32=>1)
    )
    # opt_state = Flux.setup(Adam(learning_rate), model)
    opt_state = Flux.setup(Descent(learning_rate), model)

    losses = Float32[]
    models = [deepcopy(model)]

    n_minibatches = length(y_data) ÷ minibatch_size

    @progress for epoch in 1:n_epochs
        batch_loss = zero(Float32)

        for i in 1:n_minibatches
            idxs = (1:minibatch_size) .+ minibatch_size * (i - 1)
            x_minibatch = x_data[:, idxs]
            y_minibatch = y_data[:, idxs]

            function minibatch_objective(model)
                return loss(model, x_minibatch, y_minibatch)
            end
```

$\ell(x_0, y_0)$ $\nabla_\theta \ell$

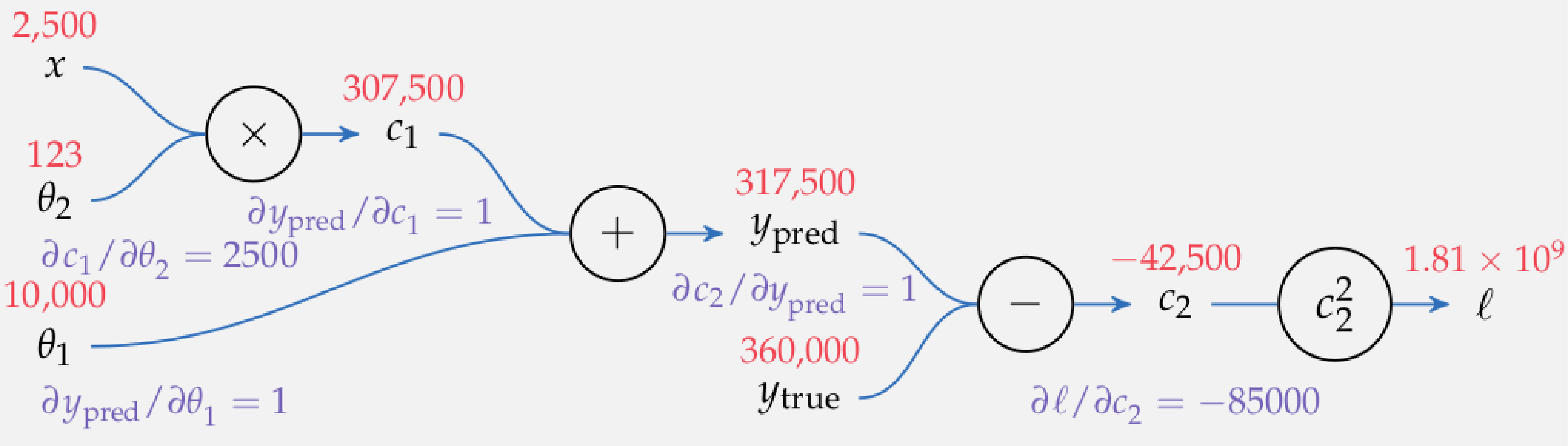
```julia
            minibatch_loss, grads = Flux.withgradient(minibatch_objective, model)

            Flux.update!(opt_state, model, grads[1])
```

$\theta \leftarrow \theta - \alpha \nabla_\theta \ell$

```julia
            batch_loss += minibatch_loss / n_minibatches
        end

        push!(losses, batch_loss)

        if epoch % save_every == 0
            push!(models, deepcopy(model))
        end
    end

    return models, losses
end
```

# Backprop

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




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

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

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

Keep calm and

lower your learning rate

# Adaptive Step Size: RMSProp

# Adaptive Step Size: ADAM

(Adaptive Moment Estimation)

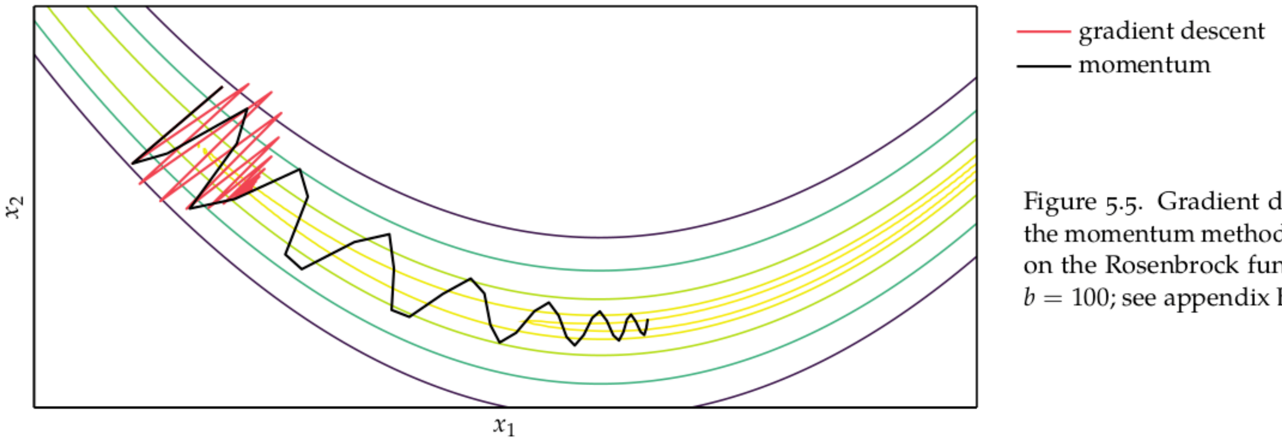



Figure 5.5. Gradient descent and the momentum method compared on the Rosenbrock function with $b = 100$; see appendix B.6.

# Adaptive Step Size: ADAM

(Adaptive Moment Estimation)

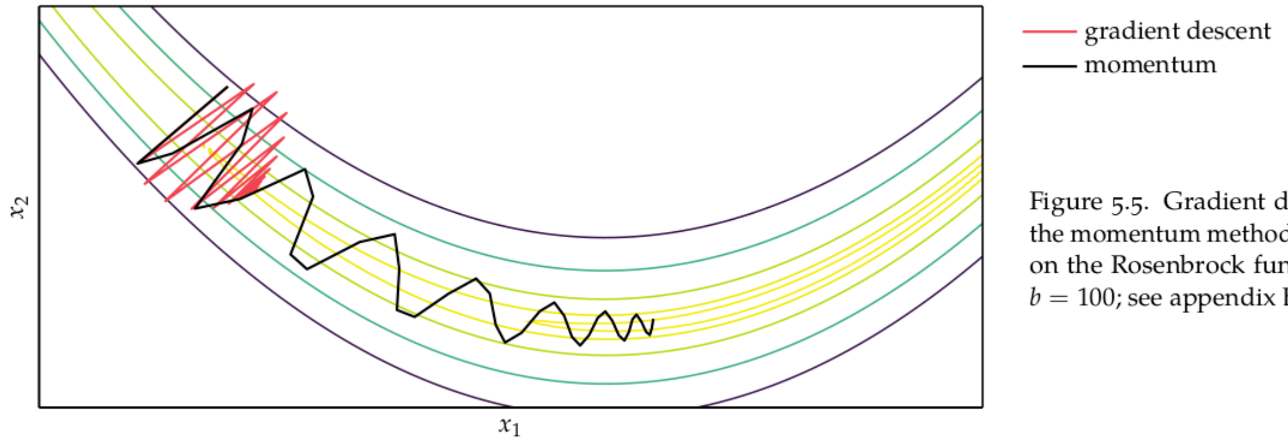



Figure 5.5. Gradient descent and the momentum method compared on the Rosenbrock function with $b = 100$; see appendix B.6.

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

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

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

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

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

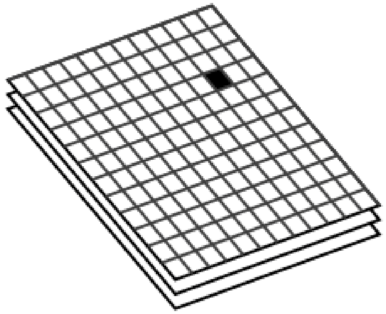
$$\theta \leftarrow \theta - \alpha \nabla_\theta \ell$$

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

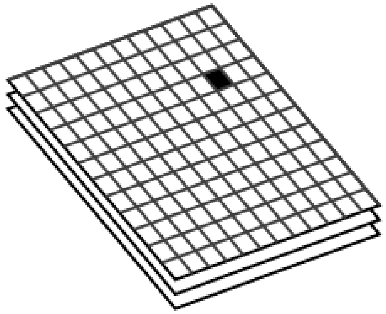
$\odot$ means elementwise multiplication.

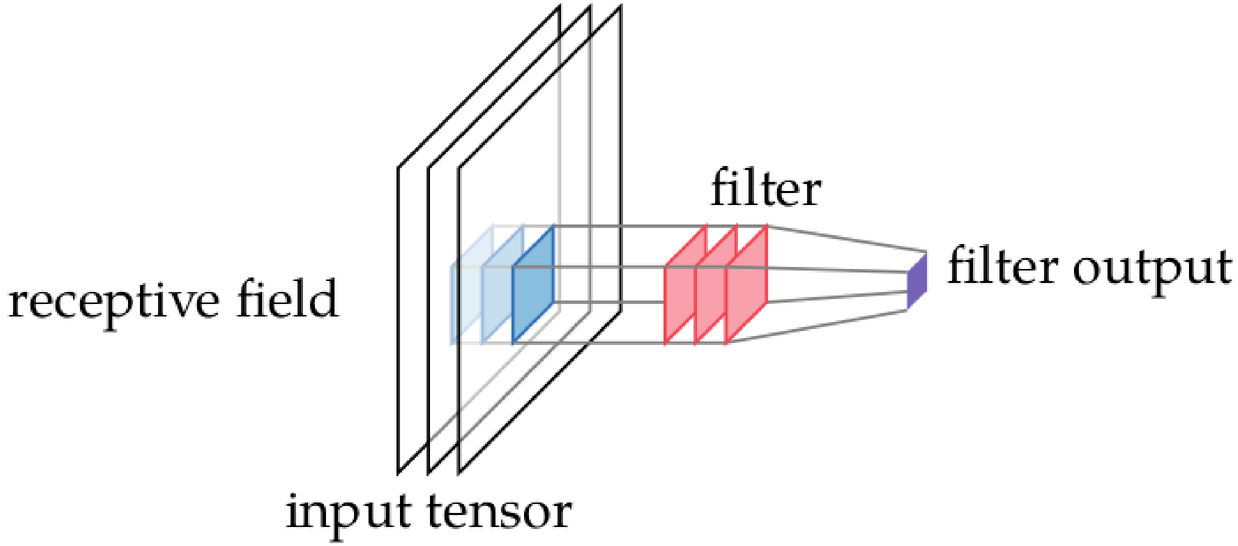
# On Your Radar: ConvNets

# On Your Radar: ConvNets

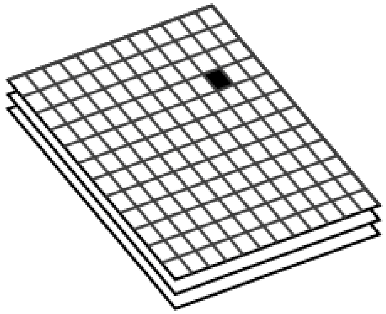


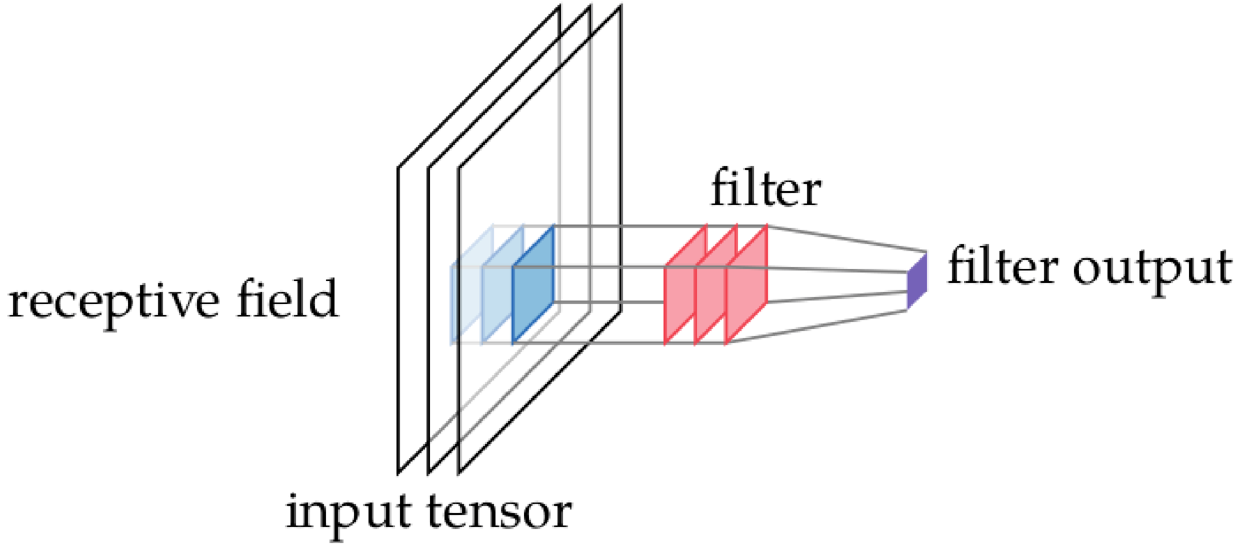
# On Your Radar: ConvNets

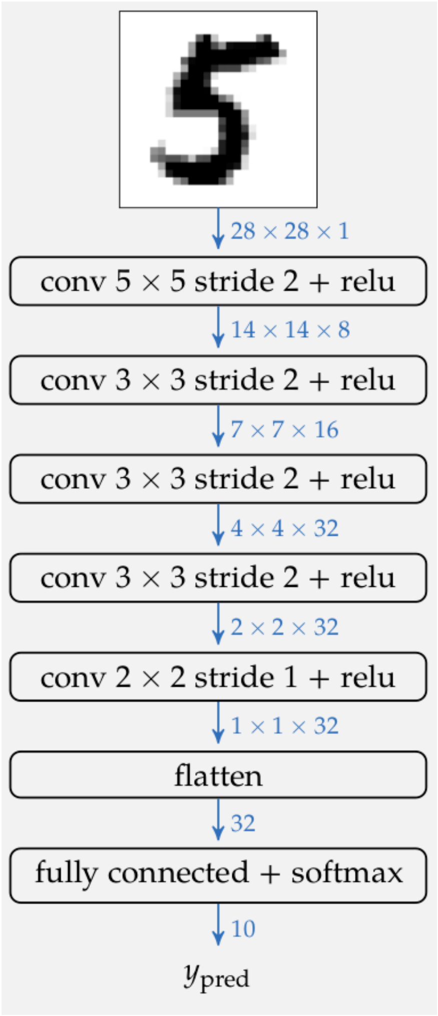




# On Your Radar: ConvNets





# On Your Radar: Regularization

# On Your Radar: Regularization

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

# 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, layer norm, dropout

# On Your Radar: Skip Connections (Resnets)

# Resources

OpenAI Spinning up