Neural Network Optimisation

Gradient-based optimisation can be broken into two steps:

- Computation of the (stochastic) gradient $\nabla_\theta L(\theta_t; \mathcal{D}_m)$
- Use of the gradient to update the parameters

The first step is the focus of the **backpropagation** algorithm, and this is implemented ‘behind the scenes’ by Keras

We have so far been using SGD for the second step, but there are several other developed optimisation algorithms that are usually more efficient than SGD

Stochastic Gradient Descent

```
for t in range(num_iterations):
```

- Sample a minibatch $\mathcal{D}_m$
- Loss function $L(\theta_t; \mathcal{D}_m) = \frac{1}{M} \sum_{x_i, y_i \in \mathcal{D}_m} l(y_i, f_{\theta_t}(x_i))$
- Gradient $\nabla_\theta L(\theta_t; \mathcal{D}_m)$
- Update rule:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t; \mathcal{D}_m)$$

Robbins, H. and Monro, S. (1951), "A stochastic approximation method", *The annals of mathematical statistics*, 400–407

Stochastic Gradient Descent with momentum

```
for t in range(num_iterations):
```

- Sample a minibatch $\mathcal{D}_m$
- Loss function $L(\theta_t; \mathcal{D}_m) = \frac{1}{M} \sum_{x_i, y_i \in \mathcal{D}_m} l(y_i, f_{\theta_t}(x_i))$
- Gradient $\nabla_\theta L(\theta_t; \mathcal{D}_m)$
- Update rule:

$$\mathbf{g}_t = \nabla_\theta L(\theta_t; \mathcal{D}_m)$$

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + \eta \mathbf{g}_t \qquad \text{(usually } \beta \approx 0.9\text{)}$$

$$\theta_{t+1} = \theta_t - \mathbf{v}_{t+1}$$

Qian, N. (1999), "On the momentum term in gradient descent learning algorithms", Neural Networks: The Official Journal of the International Neural Network Society, **12** (1), 145–151

Stochastic Gradient Descent with Nesterov momentum

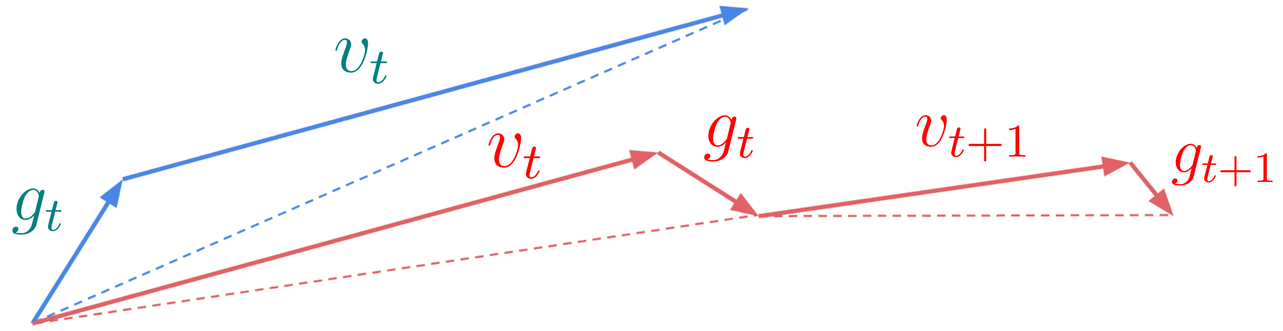
```
for t in range(num_iterations):
```

- Sample a minibatch $\mathcal{D}_m$
- Loss function $L(\theta_t; \mathcal{D}_m) = \frac{1}{M} \sum_{x_i, y_i \in \mathcal{D}_m} l(y_i, f_{\theta_t}(x_i))$
- Gradient $\nabla_\theta L(\theta_t; \mathcal{D}_m)$
- Update rule:

$$\mathbf{g}_t = \nabla_\theta L(\theta_t - \beta \mathbf{v}_t; \mathcal{D}_m)$$

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + \eta \mathbf{g}_t$$

$$\theta_{t+1} = \theta_t - \mathbf{v}_{t+1}$$



Blue: SGD with momentum
Red: SGD with Nesterov momentum

Nesterov, Y. (1983), "A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)", Doklady ANSSSR (translated as Soviet. Math. Docl.), **269**, 543–547

RMSprop

Update rule:

$$\mathbb{E}[\mathbf{g}^2]_t = \rho \mathbb{E}[\mathbf{g}^2]_{t-1} + (1-\rho)(\nabla_\theta L(\theta_t; \mathcal{D}_m))^2 \qquad \text{(usually } \rho \approx 0.9\text{)}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[\mathbf{g}^2]_t + \epsilon}} \odot \nabla_\theta L(\theta_t; \mathcal{D}_m)$$

http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Adam

Update rule:

$$\mathbb{E}[\mathbf{g}]_t = \beta_1 \mathbb{E}[\mathbf{g}]_{t-1} + (1-\beta_1)\nabla_\theta L(\theta_t; \mathcal{D}_m)$$

$$\mathbb{E}[\mathbf{g}^2]_t = \beta_2 \mathbb{E}[\mathbf{g}^2]_{t-1} + (1-\beta_2)(\nabla_\theta L(\theta_t; \mathcal{D}_m))^2$$

$$\mathbf{m}_t = \frac{\mathbb{E}[\mathbf{g}]_t}{(1-\beta_1)}$$

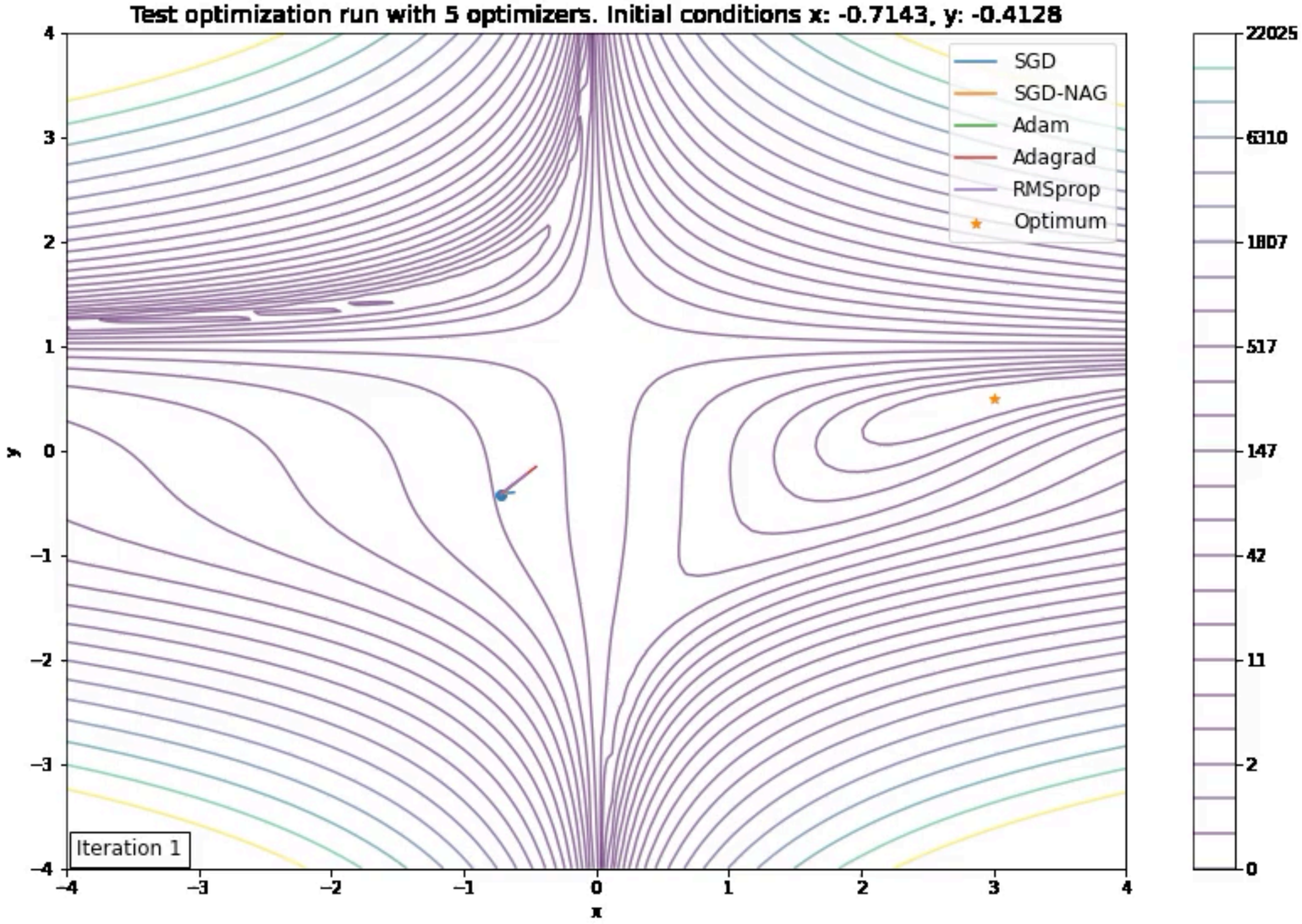
$$\mathbf{v}_t = \frac{\mathbb{E}[\mathbf{g}^2]_t}{(1-\beta_2)}$$

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

(usually $\beta_1 \approx 0.9$, $\beta_2 \approx 0.999$)

Kingma, D. P. & Ba, J. L. (2015), "Adam: a Method for Stochastic Optimization", International Conference on Learning Representations, 1–13

Optimisers demo



Regularisation

Regularisation methods aim to constrain the model capacity in some way.

- Control model complexity
- Weight sharing
- Dataset augmentation
- Dropout
- Weight regularisation / weight decay
- Patience / early stopping

Weight Regularisation

Adds a small penalty term to the loss function

$$f(\mathbf{x}) = \sum_j w_j \phi_j(\mathbf{x})$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - f_{\mathbf{w}}(x_i))^2 + \frac{\lambda}{2} \sum_j w_j^2 \right\}$$

Weight decay: $\mathbf{w}_{t+1} = (1 - \lambda)\mathbf{w}_t - \eta \mathbf{g}_t$

Hanson, S. J. & Pratt, L. Y. (1988) "Comparing biases for minimal network construction with back-propagation", in *Proceedings of the 1st International Conference on Neural Information Processing Systems*, 177–185.

Dropout

Randomly drops units and connections in a neural network

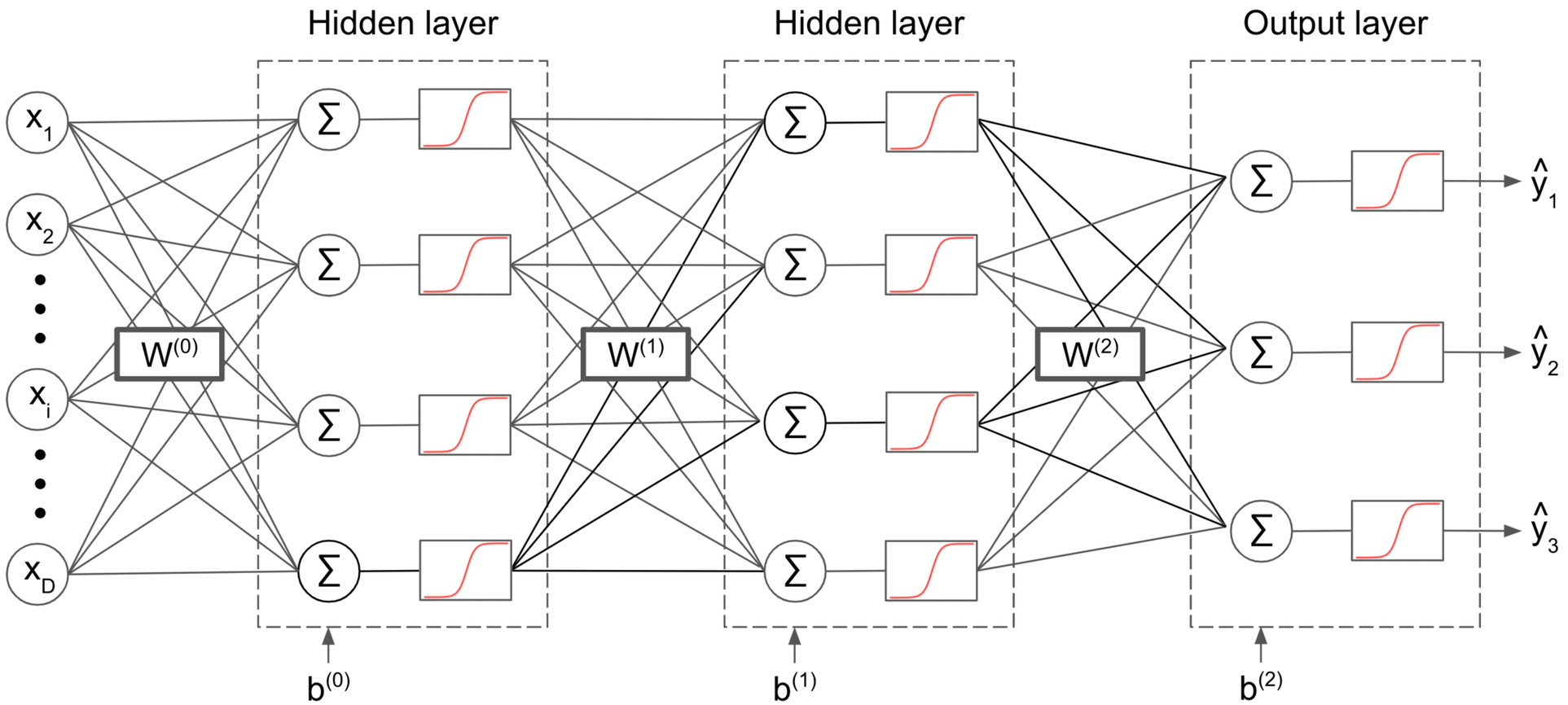
$$\mathbf{h}^{(k+1)} = \sigma\left(\mathbf{W}^{(k)}\mathbf{h}^{(k)} + \mathbf{b}^{(k)}\right)$$

$$\mathbf{W}^{k} \leftarrow \mathbf{W}^{k} \cdot \mathrm{diag}([\mathbf{z}_{k,j}]_{j=1}^{n_{k-1}})$$

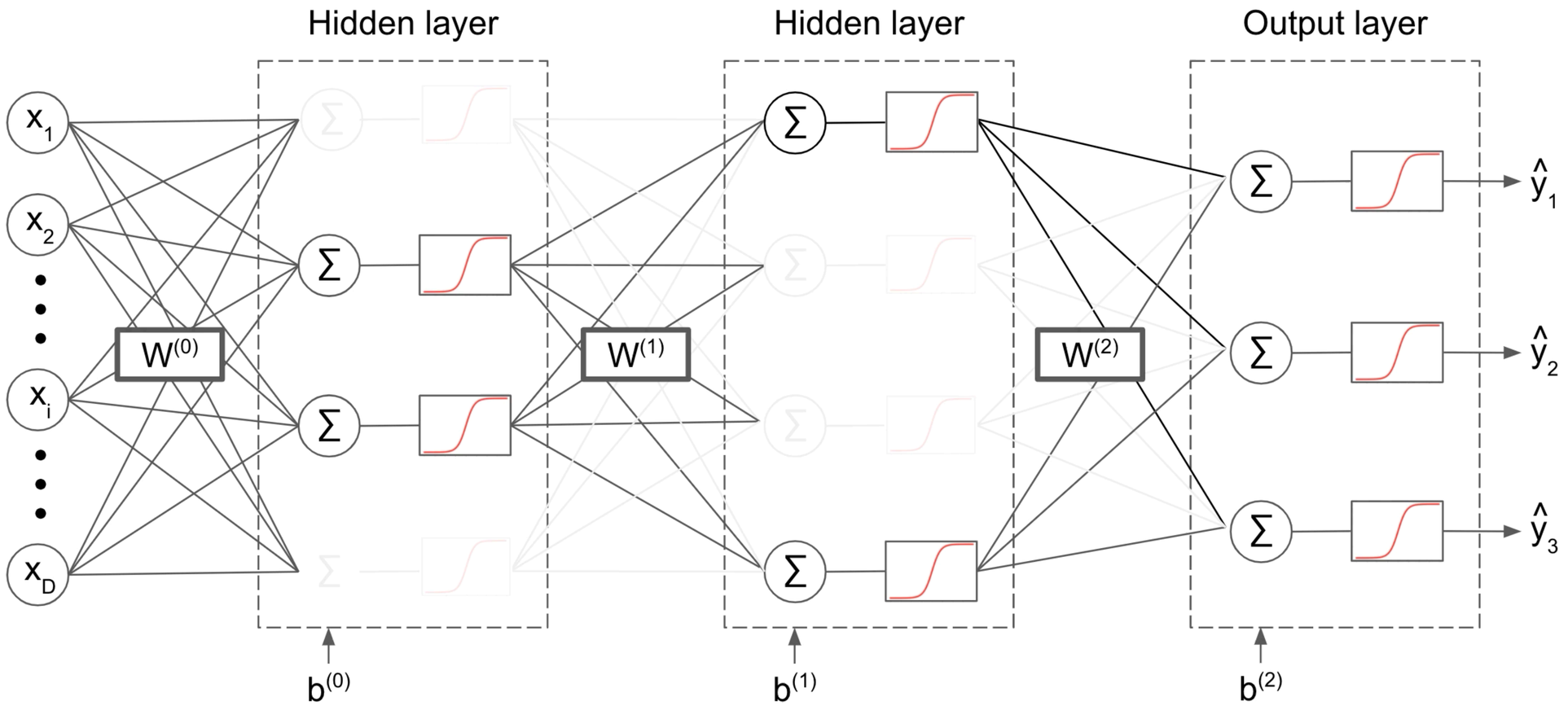
$$\mathbf{z}_{k,j} \sim \mathrm{Bernoulli}(p_k), \qquad k = 1, \ldots, L$$

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014), "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research, **15**, 1929-1958

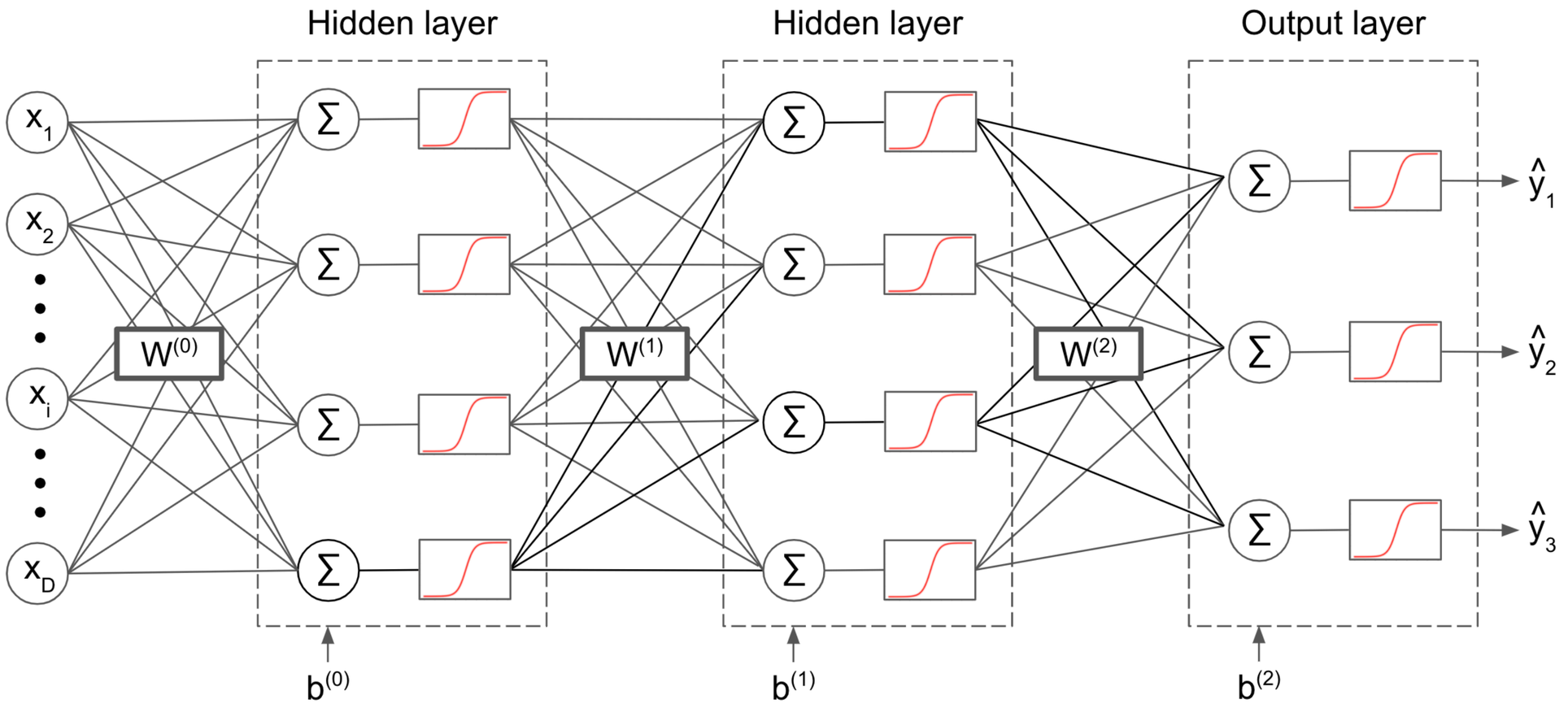
Dropout


Hidden layer
Hidden layer
Output layer
x_1
x_2
x_i
x_D
$W^{(0)}$
$W^{(1)}$
$W^{(2)}$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$b^{(0)}$
$b^{(1)}$
$b^{(2)}$

Dropout


Hidden layer
Hidden layer
Output layer
x_1
x_2
x_i
x_D
$W^{(0)}$
$W^{(1)}$
$W^{(2)}$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$b^{(0)}$
$b^{(1)}$
$b^{(2)}$

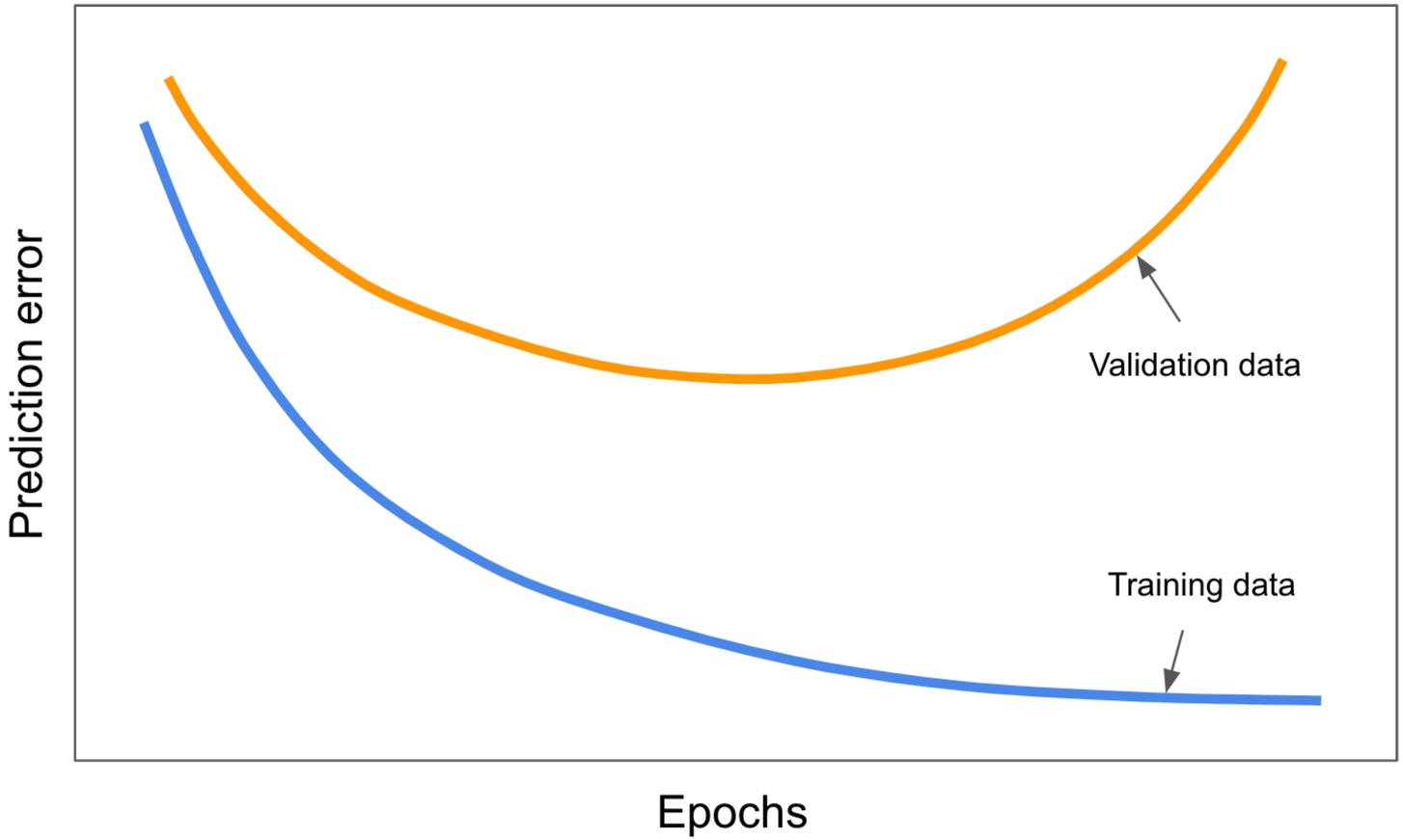
Dropout



After training: scale down weights by multiplying by the 'keep rate': $\mathbf{W}^{(k)} \leftarrow \mathbf{W}^{(k)} \cdot p_k$

Early Stopping / Patience

Constrains capacity by limiting the training time



Early Stopping / Patience

```
best_valid_loss = np.inf
patience = 0

for epoch in range(max_epochs):
   epoch_train_loss = train_model(train_data, train_loss)
   epoch_valid_loss = validate_model(valid_data, val_metric)
   if epoch_valid_loss < best_valid_loss:
       best_valid_loss = epoch_valid_loss
       patience = 0
   else:
       patience += 1

   save_model(epoch)

   if patience >= max_patience:
       break  # terminate training
```