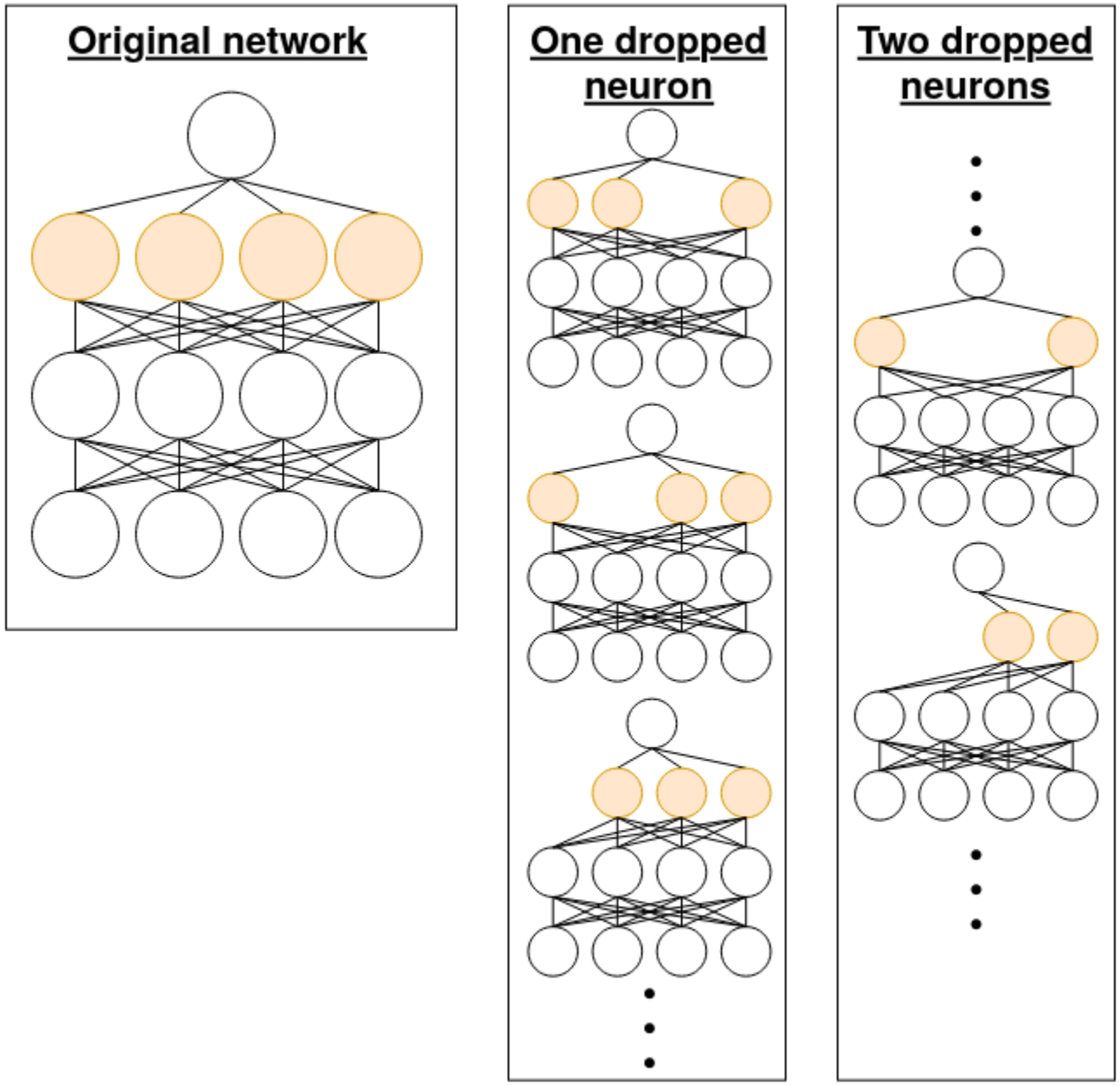
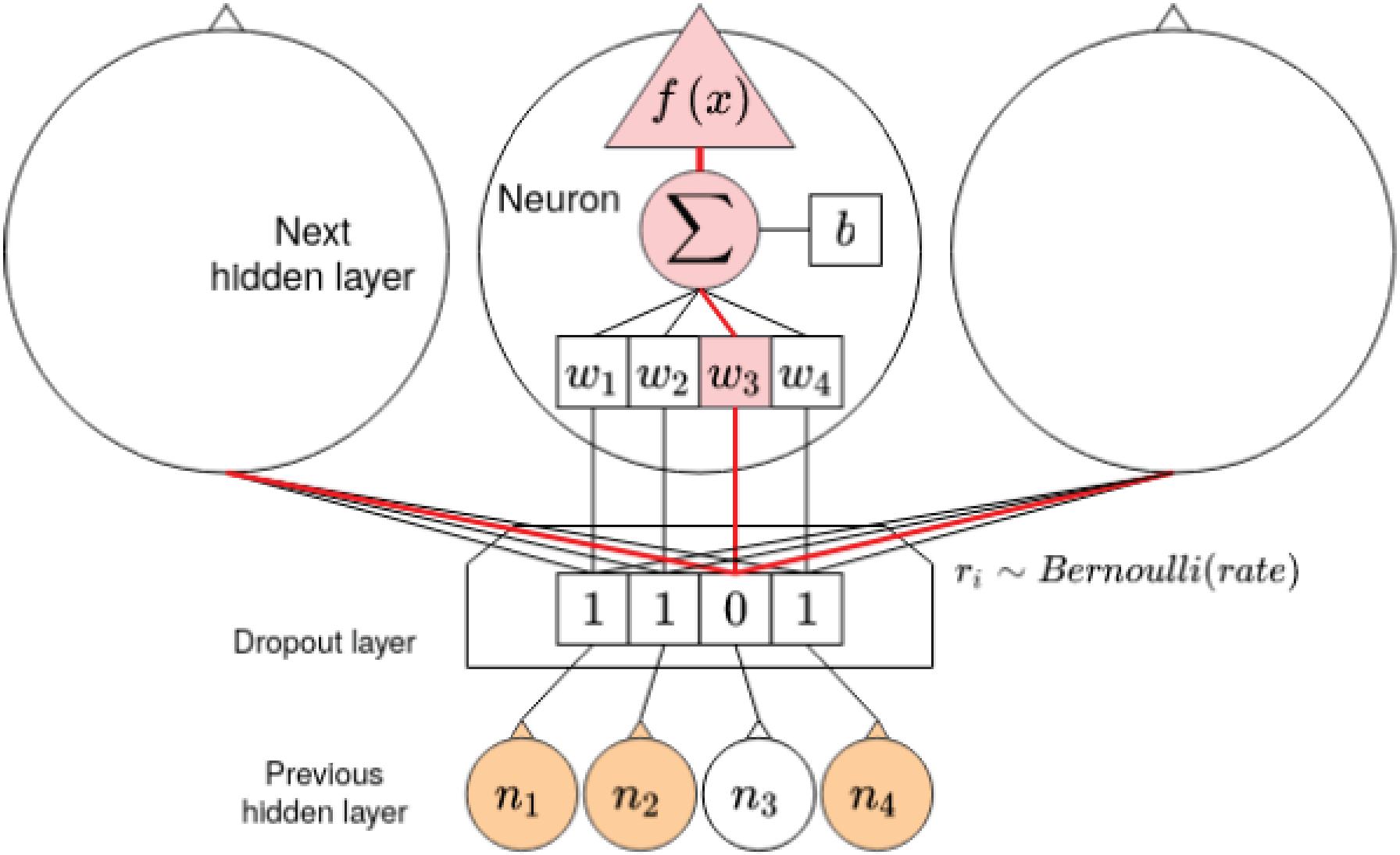
Dropout

- Temporarily *removing* (dropping out) *some* input or hidden *neurons during* network *training*
- Neurons are dropped out *randomly*, according to a given distribution
- Originally proposed for and most often used during training of multilayer perceptrons


Original network
One dropped neuron
Two dropped neurons

Bernoulli dropout

- Bernoulli(p) distribution: on $\{a, b\}$ with probabilities $(1-p, p)$
- Assumptions about l-th *hidden layer*, $l = 1, \ldots, L$:
  - vectorial input $z^{(l)}$, output $y^{(l)}$, weight $w^{(l)}$, scalar bias $b^{(l)}$
  - activation function f does not depend on l, relates $y^{(l)} = f\left(z^{(l)}\right)$
  - in addition: set $a = 0, b = 1$, denote $y^{(0)} = x$ – network input
- Then $z_i^{(l)} = w_i^{(l)} r_i^{(l)} y_i^{(l-1)} + b^{(l)}$, with *random* $r_i^{(l)} \sim$ *Bernoulli*(p)


Next
hidden layer
Neuron
$f(x)$
$\sum$
b
w_1 w_2 w_3 w_4
$r_i \sim Bernoulli(rate)$
1 1 0 1
Dropout layer
Previous
hidden layer
n_1 n_2 n_3 n_4

Dropout and network training

- Most often using stochastic gradient descent
- Difference from standard MLP: for each *training case*,

  new values $r_i^{(l)}$ are sampled $\Longrightarrow$ a new *specific network*

  - forward- and backpropagation restricted to that individual network
- *Gradients* are *averaged* over cases retaining the parameter

  - cases with that parameter dopped out $\Longrightarrow$ gradient contribution = 0

Dropout and regularization

- Dropout alone improves training, with regularization even more
- Most often combined with *max-norm regularization:* $\|w\| \leq c$
  - w – vector of all weights, $\|\ \ \|$ – some norm, c – hyperparameter
  - $\Longrightarrow$ network learning is then constrained optimization
- Main reason why max-norm regularization is useful:

  no weigths blowup through large learning rate $\Longrightarrow$ explorability

Some other properties of dropout

- *Sparse representation*, even if no sparsity inducing regularizers
- *Influence of dataset size* relatively to network size:
  - very small datasets overfitting even after dropout $\Longrightarrow$ useless
  - with increasing dataset size, its usefulness increases, then again decreases $\Longleftarrow$ for very large datasets, no overfitting occurs
- Training time: $2 - 3\times$ longer than withouf dropout

Advantages of dropout

1. After dropout, the network has less parameters $\Longrightarrow$

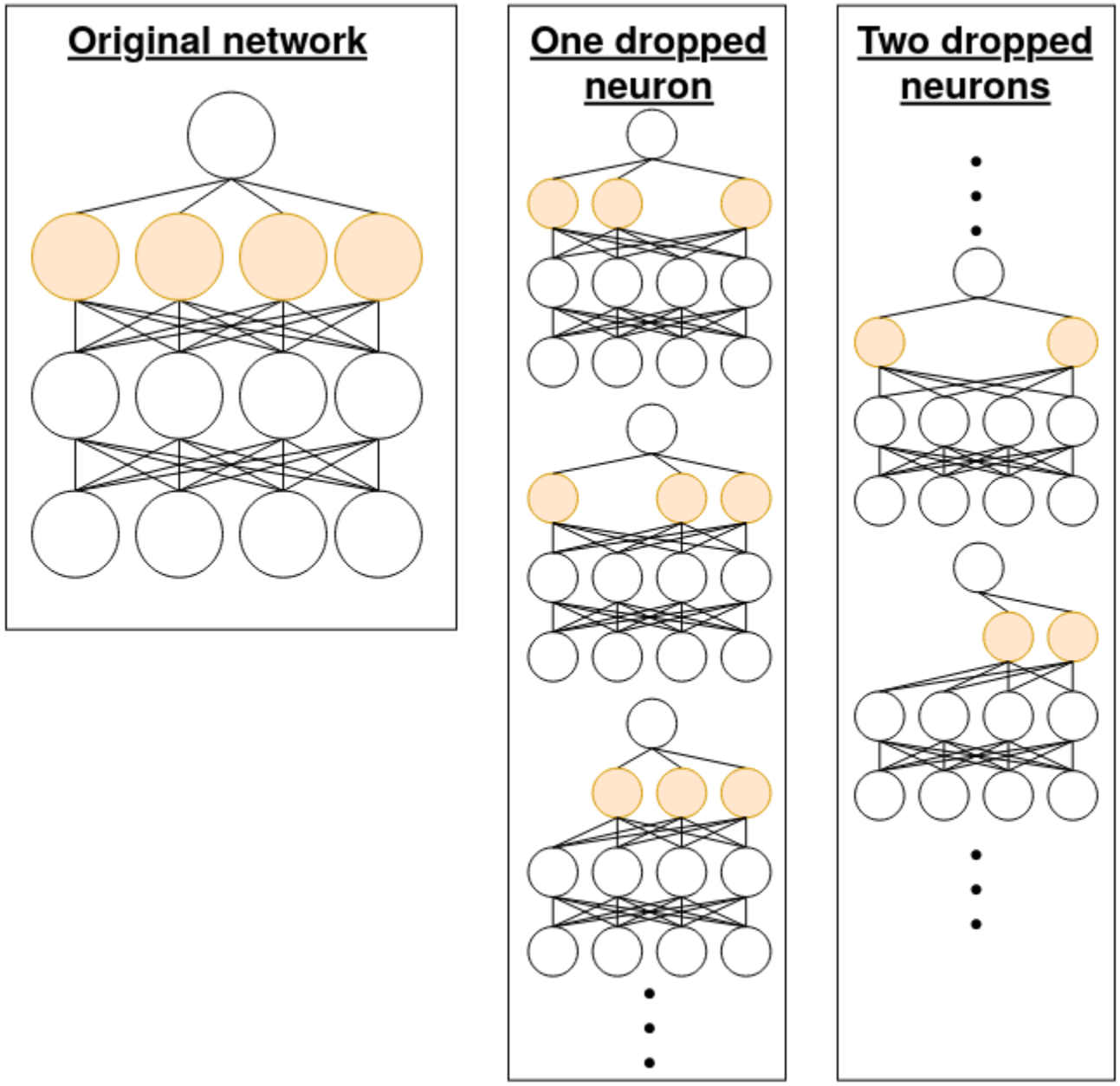
   $\Longrightarrow$ less prone to *overfitting* the training data

2. Breaking-up co-adaptations of different hidden neurons,

   which impede generalization $\Longrightarrow$ improved *generalization*

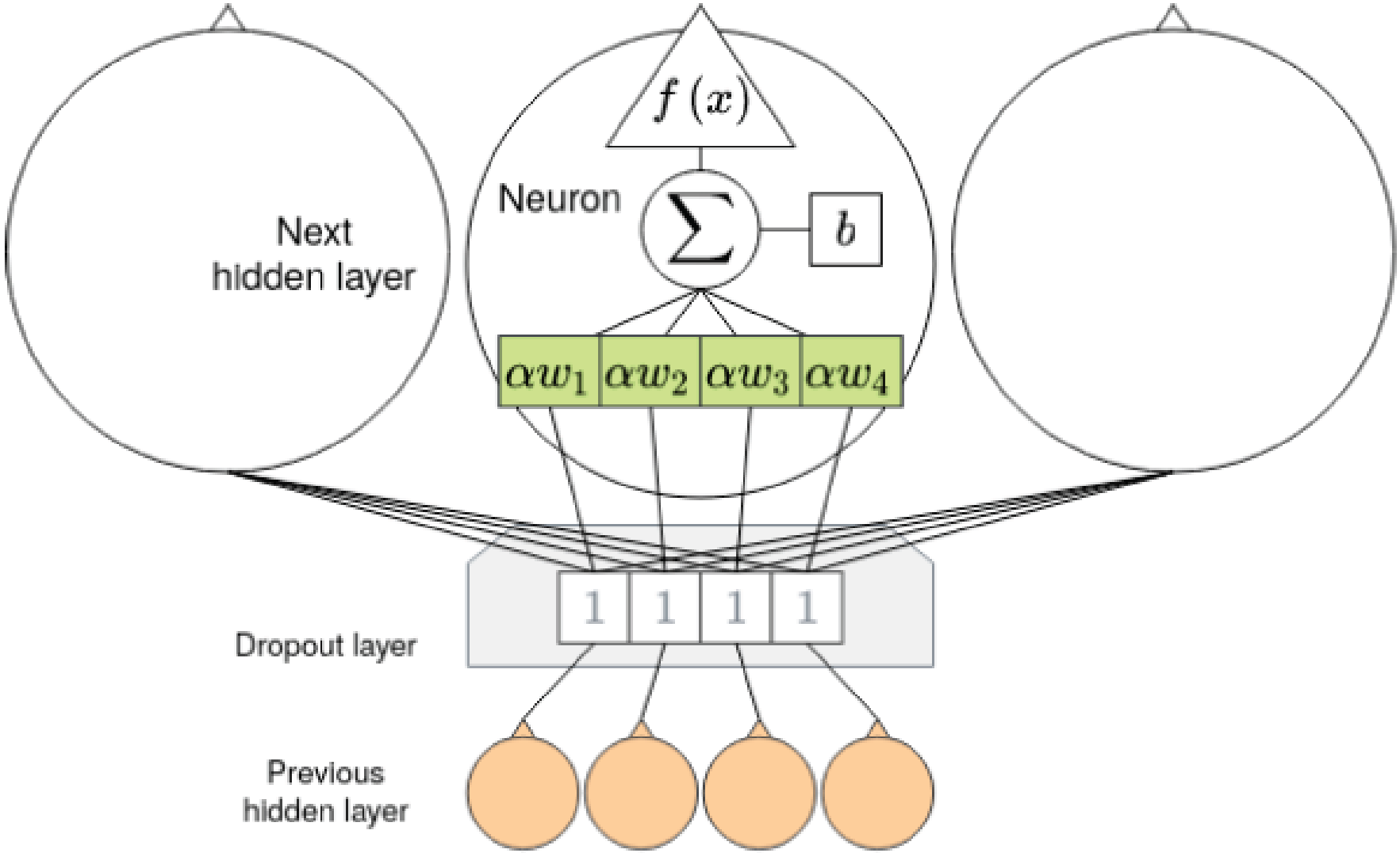
3. Different dropout realizations $\approx$ different network topologies $\Longrightarrow$

   $\Longrightarrow$ dropout implies building network *ensembles*


Original network
One dropped neuron
Two dropped neurons

Dropout ensembles

- For an ensemble S built through dropping out subsets of the set H of hidden neurons: $|\mathcal{S}| \leq 2^{|H|}$
- If during training, $h \in H$ survives dropout with probability p, then during testing, weights outgoing from h are multiplied by p
  - $\Rightarrow$ *expected* weights *after training = used testing weights*
- Alternative possibility: training weights multilplied by $\frac{1}{p}$


$f(x)$
Neuron
$\sum$
b
Next
hidden layer
αw_1 αw_2 αw_3 αw_4
1 1 1 1
Dropout layer
Previous
hidden layer

More general dropouts

- Used also with other models than multilayer perceptrons
  - *restricted Boltzmann* machine (RBM, will be described later)
  - *linear regression* (will be described later)
- Used also with other distributions than Bernoulli
  - *Gaussian distribution* (will be described later)

Introducing dropout into RBM

- RBM with visible units $v \in \{0,1\}^{d_v}$, hidden units $h \in \{0,1\}^{d_h}$ and parameters $\theta = (W, a, b), W \in \mathbb{R}^{d_v \times d_h}, a \in \mathbb{R}^{d_h}, b \in \mathbb{R}^{d_v}$, which define $P(h, v; \theta) = \frac{\exp(v^\intercal W h + a^\intercal h + b^\intercal v)}{C(\theta)}$, $C(\theta)$ - normalizing constant
- *Dropout* is introduced with a $\{0,1\}^{d_h}$-valued random vector r with random components r_j~*Bernoulli(p)*, $r_j = 1 \Leftrightarrow h_j = 1$,
  - consequence: $r_j = 1 \Rightarrow h_j = 1$, $r_j = 0 \Rightarrow h_j = 0$

Dropout RBM probability distribution

- *Joint* distribution of v *and* h, with a normalizing constant $C(\theta, r)$:

$$P(h, v; \theta) = \frac{\exp\left(v^\top W h + a^\top h + b^\top v\right)}{C(\theta, r)} \prod_{j=1}^{d_h} \left(\mathbb{I}\left(r_j = 1\right) + \mathbb{I}\left(r_j = 0\right) \mathbb{I}\left(h_j = 0\right) \right)$$

- Conditional distribution of h *conditioned on* r *and* v:

$$P(h|r, v) = \prod_{j=1}^{d_h} P\left(h_j | r_j, v\right), P\left(h_j = 1 | r_j, v\right) = \mathbb{I}\left(r_j = 1\right) \sigma\left(b_j + \sum_i W_{ij} v_i\right)$$

- Conditional distribution of v *on* h (same as without dropout):

$$P(v|h) = \prod_{i=1}^{d_v} P(v_i|h), P(v_i = 1|h) = \sigma\left(a_i + \sum_i W_{ij} h_j\right)$$

Dropout in linear regression

- Dropped out are individual training pairs − rows of (X, y)
  - $X \in \mathbb{R}^{N \times d}$ − matrix of N data points, $y \in \mathbb{R}^{d}$ − vector of targets
- Dropout introduced through a component-wise product $X \odot R$
  - $R \in \{0,1\}^{N \times d}$ is a $\{0,1\}^{N \times d}$-valued random matrix
  - R has all its *components random* $R_{ij} \sim Bernoulli(p)$

Learning dropout linear regression

- Learning in traditional linear regression consists in finding a weight vector $w \in \mathbb{R}^d$ minimizing the error $\|y - Xw\|^2$
- For dropout linear regression learning, the *minimized error* turns to $\mathbb{E}_{R\sim \text{Bernoulli}(p)} \|y - X \odot R\, w\|^2 =$ (after computation)

$$= \|y - pXw\|^2 + p(1-p) \left\| \left(\text{diag}(X^\intercal X)\right)^{\frac{1}{2}} w \right\|^2 =$$

$$= \|y - X\tilde{w}\|^2 + \frac{1-p}{p} \left\| \left(\text{diag}(X^\intercal X)\right)^{\frac{1}{2}} \tilde{w} \right\|^2, \text{ with } \tilde{w} = pw$$

Gaussian dropout

- Basic idea: *activation* h_i of the hidden neuron i is perturbed to $\mathrm{h_i}(1+\mathrm{r})$ with $\mathrm{r}\sim\mathrm{N}(0,1)$, more generally $\mathrm{r}\sim\mathrm{N}(0,\sigma^2)$
- Equivalently: activation h_i is *perturbed to* $h_i r'$ with $r' = 1 + r$, hence $r' \sim N(1,1)$, more generally $r' \sim N(1,\sigma^2)$
- *Hyperparameter* σ^2*, like* p in Bernoulli dropout

What does the Gaussian drop out?

- Formally, Gaussian dropout drops no neurons out,

  only perturbs the activations of hidden neurons

- However, for $h_i r'$ with $r' \sim N(1, \sigma^2)$, where $\sigma^2 = \frac{1-p}{p}$:

  the expectation and variance of r' are $\mathbb{E}r' = 1, \mathrm{Var}\, r' = \frac{1-p}{p}$

- And the same $\mathbb{E}r'$ and $\mathrm{Var}\, r'$ has $r' \sim$ *Bernoulli*(p) on $\left\{0, \frac{1}{p}\right\}$,

  which drops out the hidden neuron i