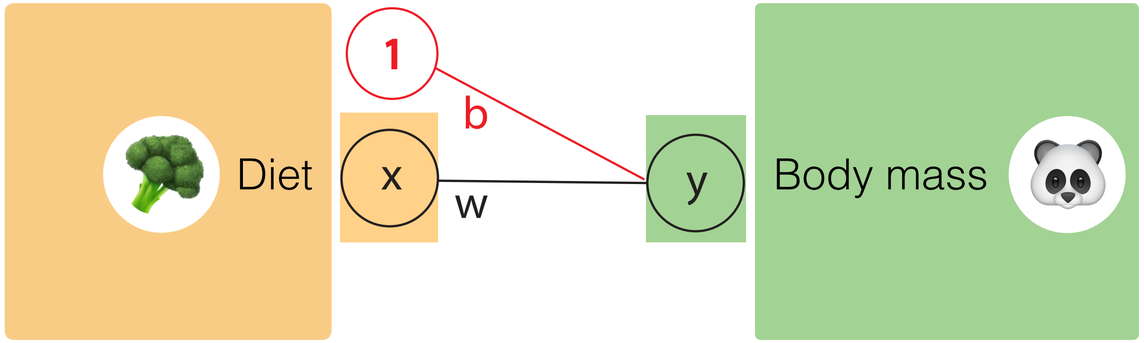
Linear regression



$$y = wx + b$$

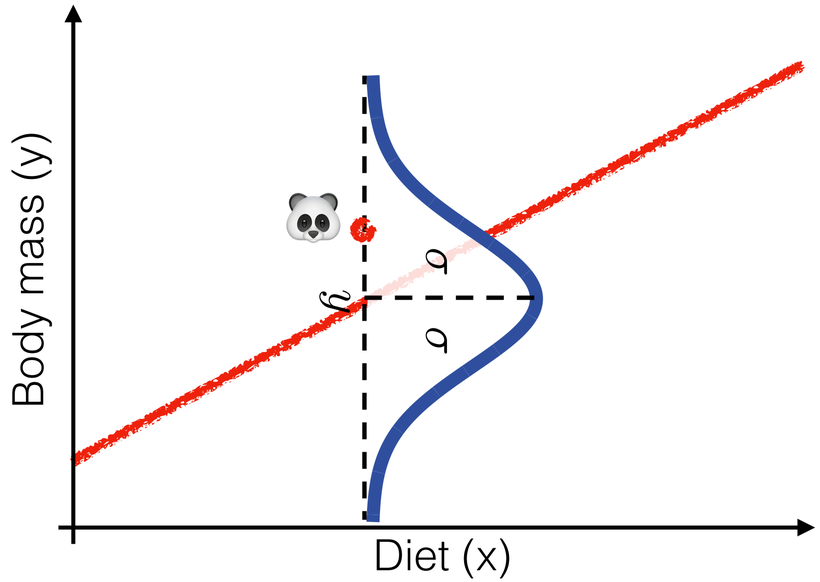
prediction slope intercept

$$🐼 = y \pm \sigma$$

error (st. dev.)

Likelihood of an observed panda body mass

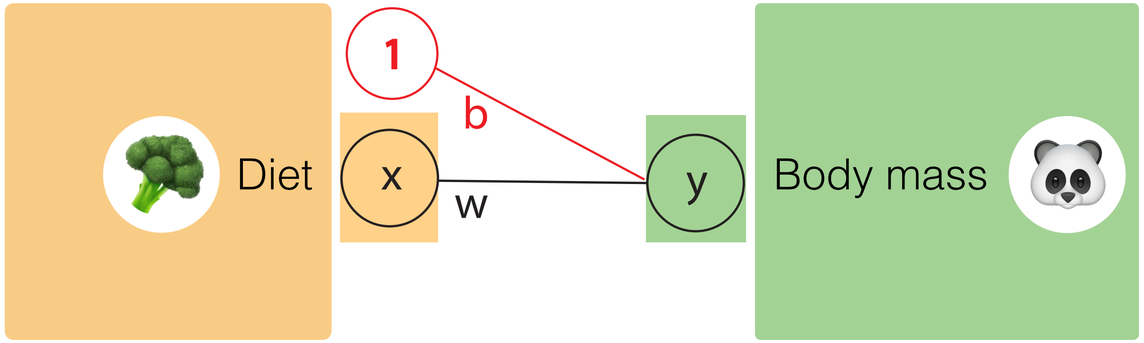
$$P(🐼 | x, w, b, \sigma) \sim \mathcal{N}(y, \sigma)$$



Probability density function of a normal distribution

$$P(🐼 \,|\, \text{y}, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(🐼 - \text{y})^2}{2\sigma^2}}$$

Linear regression



$$y = wx + b$$

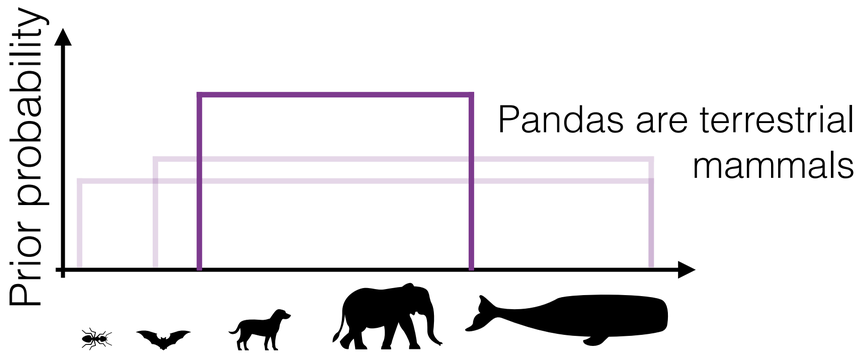
prediction — slope — intercept

$$🐼 = y \pm \sigma$$

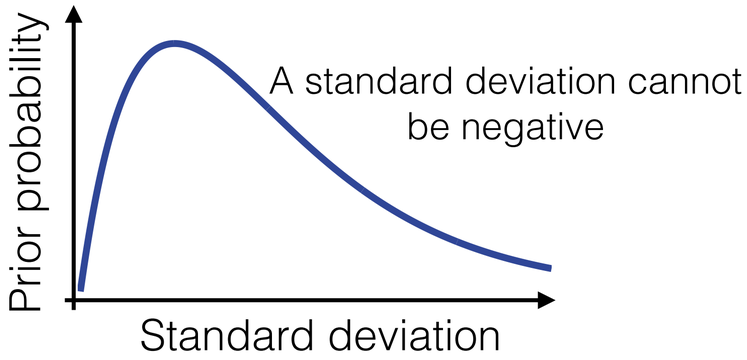
error (st. dev.)

Priors on the model parameters

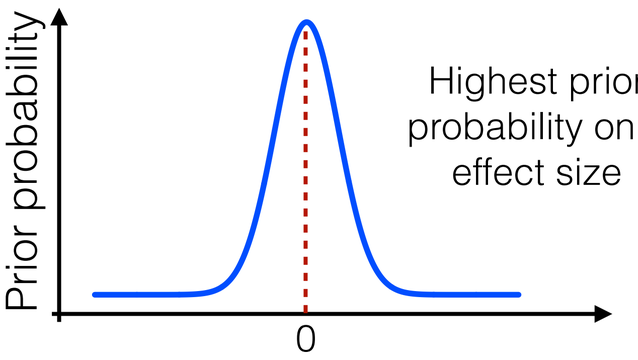
Intercept: $P(b) \sim \mathcal{U}(🐭 \mid 🦏)$



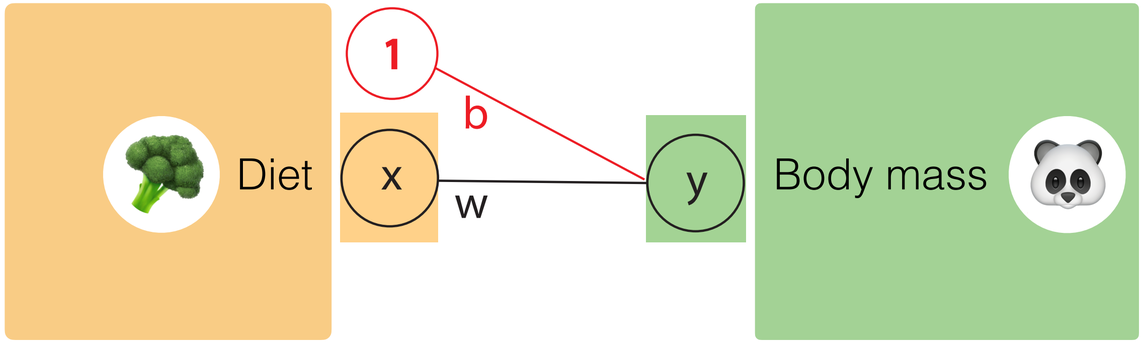
Error: $P(\sigma) \sim \Gamma(\alpha, \beta)$



Slope: $P(w) \sim \mathcal{N}(0, 1)$



Linear regression



$$y = wx + b$$

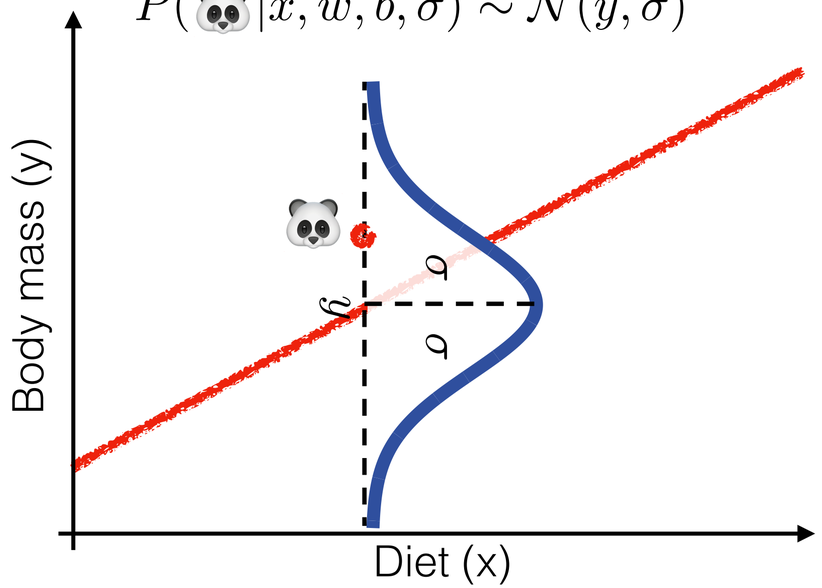
prediction slope intercept

$$\text{🐼} = y \pm \sigma$$

error (st. dev.)

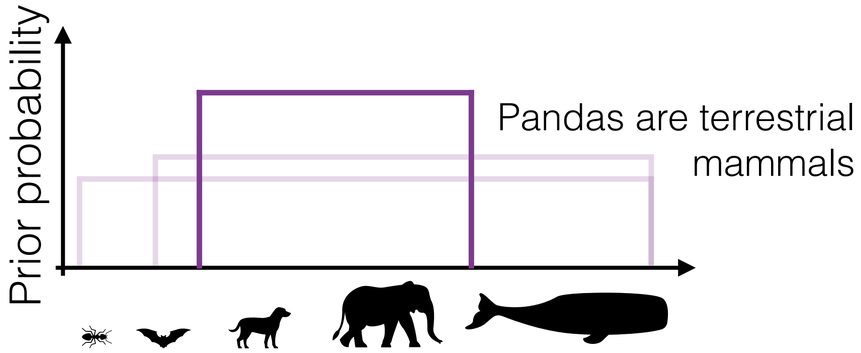
Likelihood of an observed panda body mass

$$P(\text{🐼} \mid x, w, b, \sigma) \sim \mathcal{N}(y, \sigma)$$

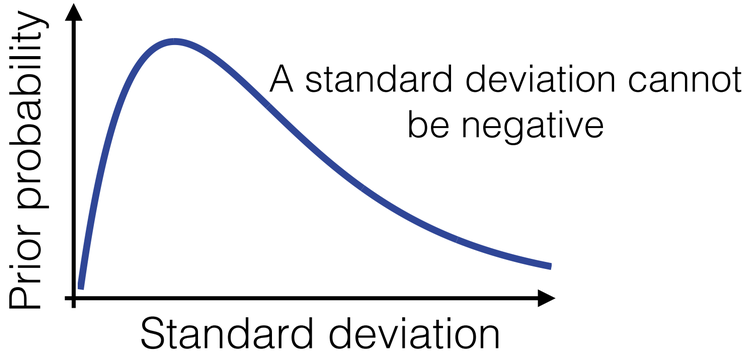


Priors on the model parameters

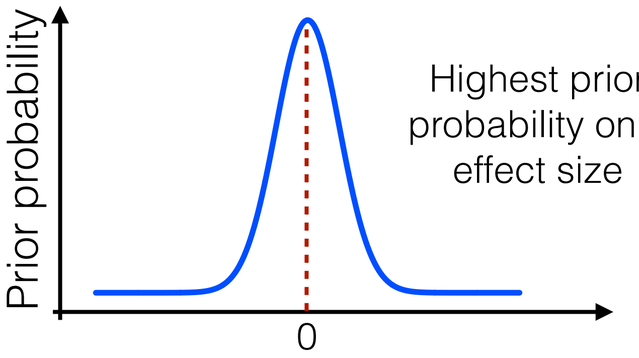
Intercept: $P(b) \sim \mathcal{U}(\text{🐭} \mid \text{🦏})$



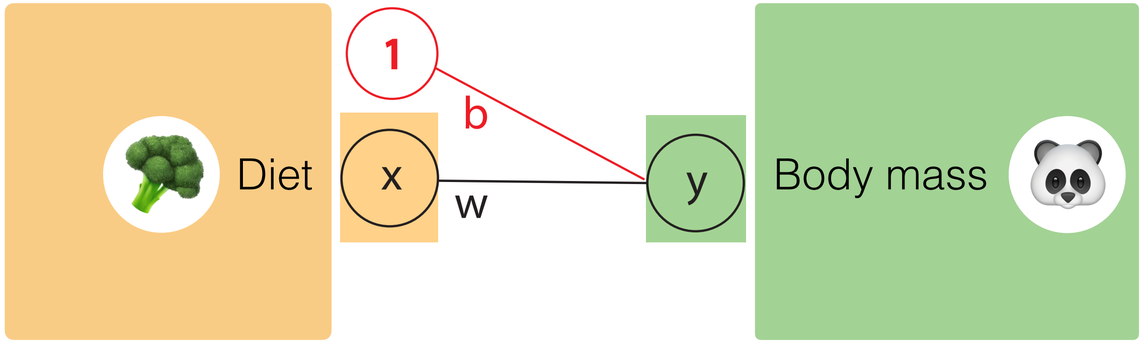
Error: $P(\sigma) \sim \Gamma(\alpha, \beta)$



Slope: $P(w) \sim \mathcal{N}(0, 1)$



Linear regression



$$y = wx + b$$

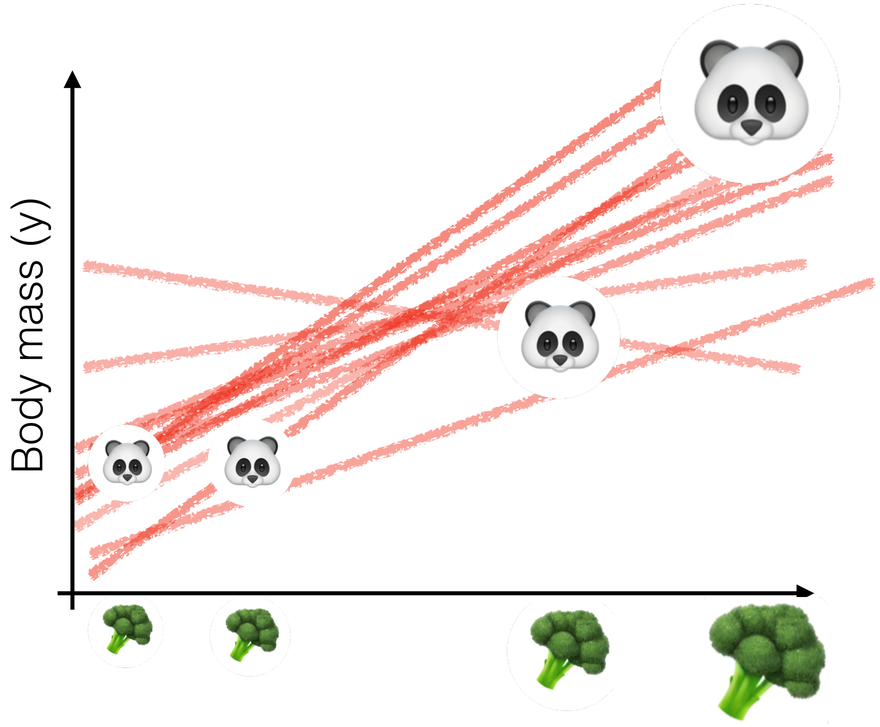
prediction slope intercept

$$\text{🐼} = y \pm \sigma$$

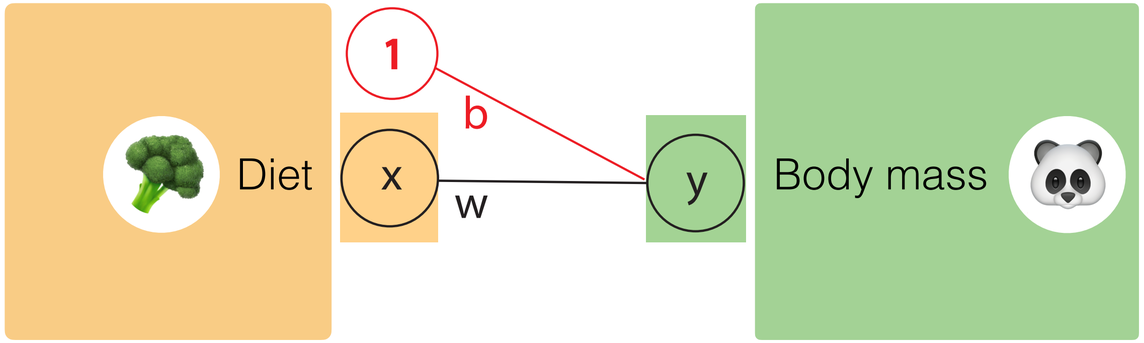
error (st. dev.)

Data (‘training set’)

Optimizing (‘training’) a model



Linear regression



$$y = wx + b$$

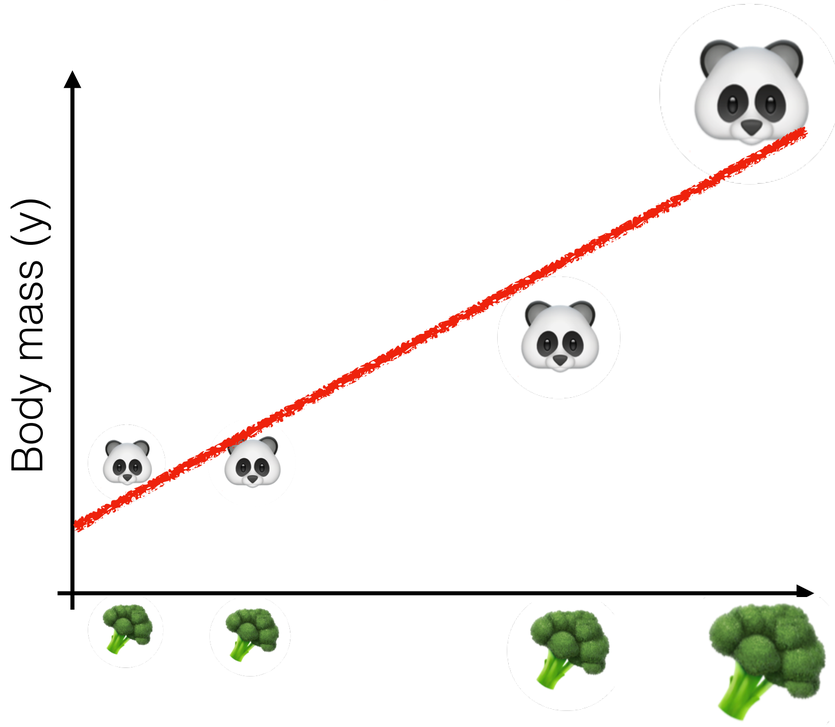
prediction slope intercept

$$\text{🐼} = y \pm \sigma$$

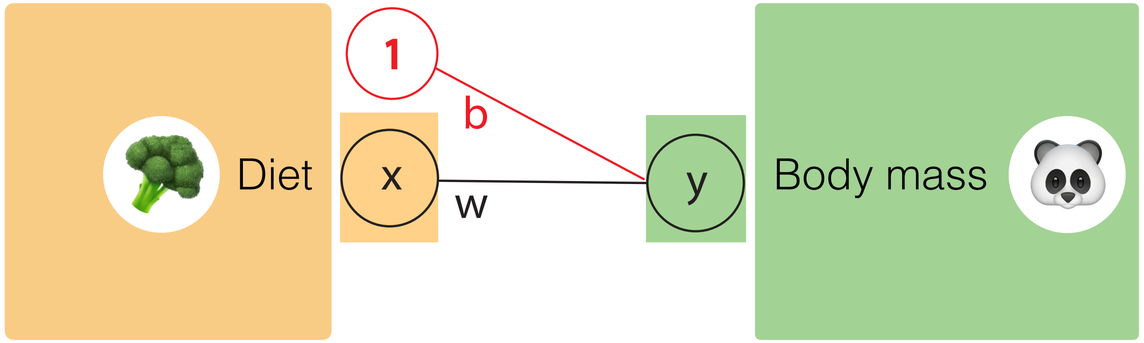
error (st. dev.)

Data (‘training set’)

Estimated (‘trained’) model



Linear regression



$$y = wx + b$$

prediction slope intercept

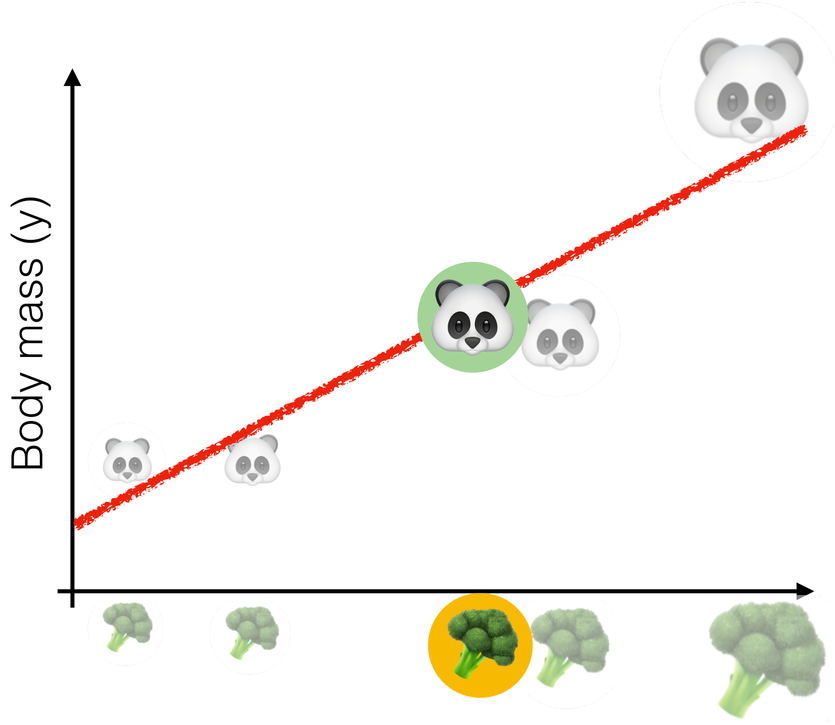
$$\text{🐼} = y \pm \sigma$$

error (st. dev.)

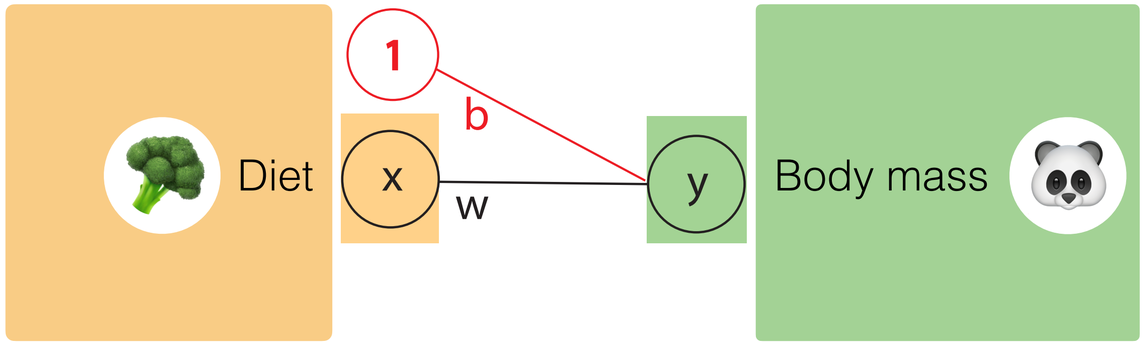
Predicting body mass from diet

Data (‘training set’)

Estimated (‘trained’) model

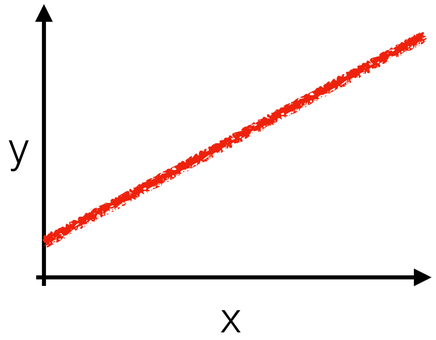


Linear regression

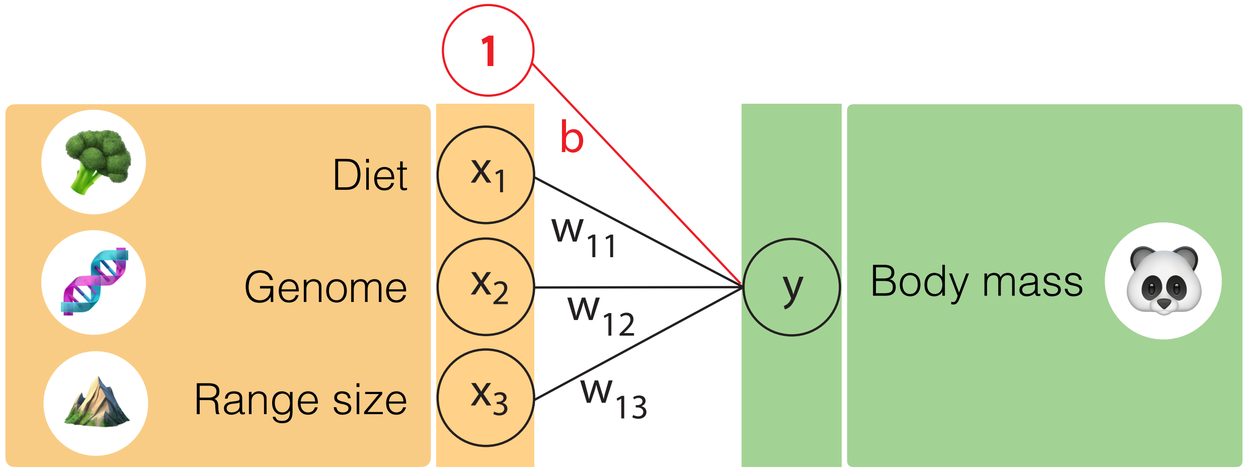


$$y = wx + b$$

prediction — slope — intercept



Multiple linear regression



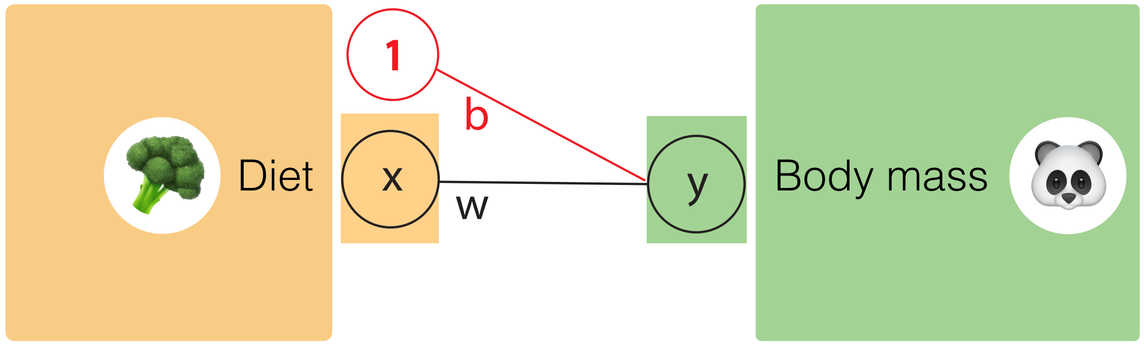
$$y = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b$$

prediction — slopes (effect sizes) — intercept

$$y = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \end{pmatrix} + b$$

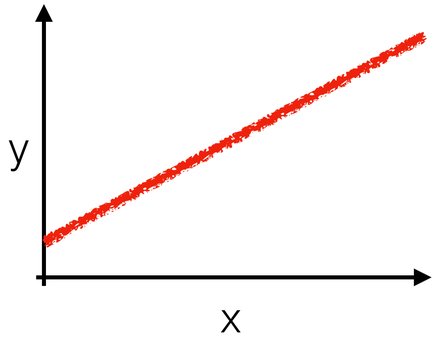
We can re-write it as a matrix multiplication

Linear regression

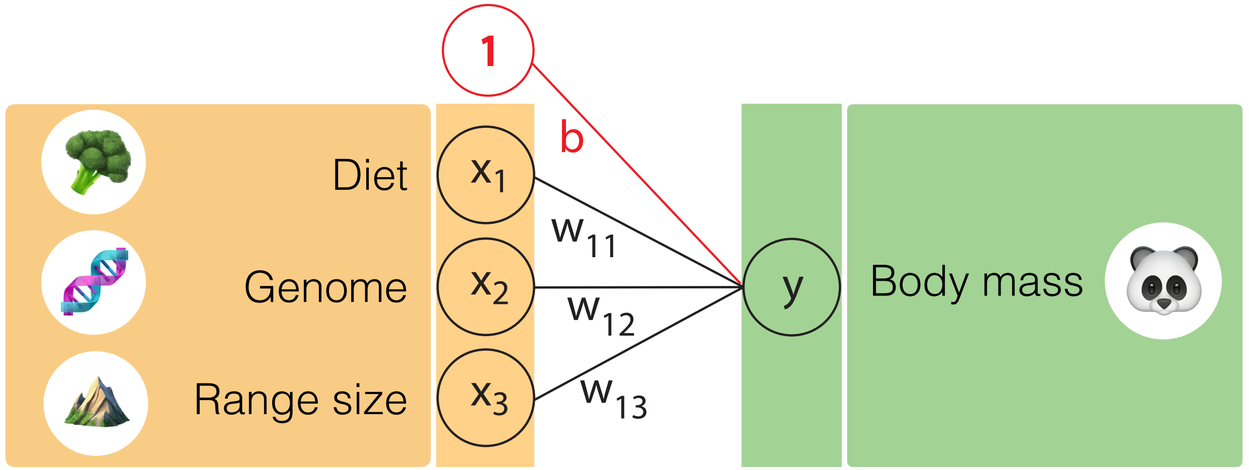


$$y = wx + b$$

prediction — slope — intercept



Multiple linear regression



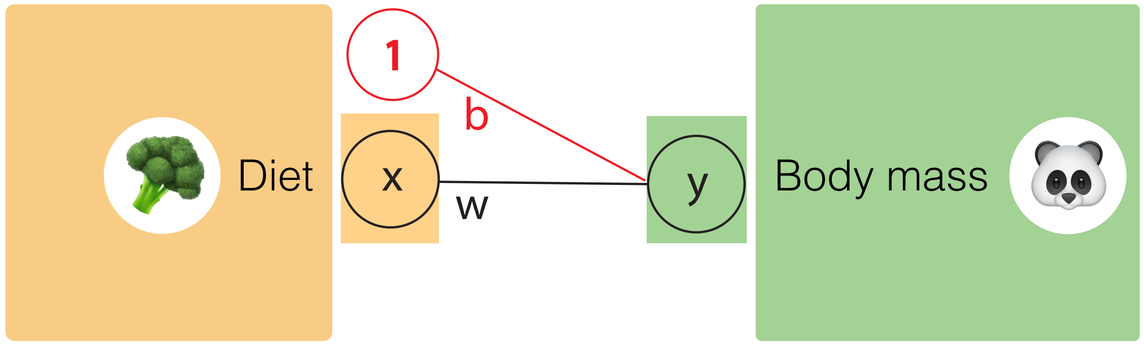
$$y = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b$$

prediction — slopes (effect sizes) — intercept

$$y = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \end{pmatrix} + b$$

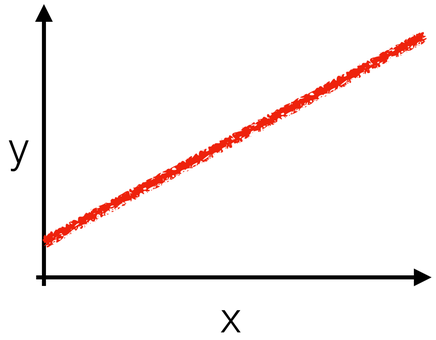
We can re-write it as a matrix multiplication

Linear regression

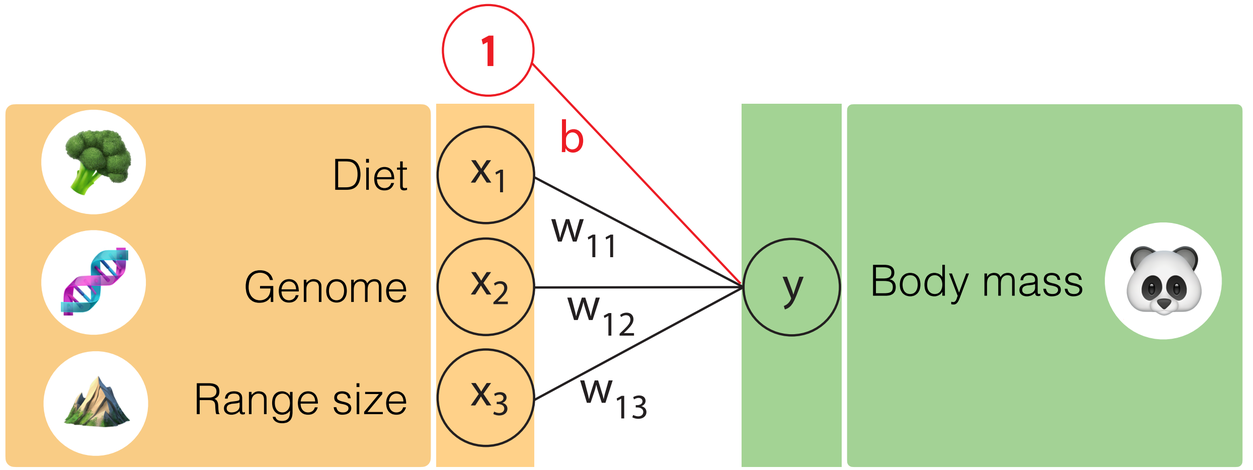


$$y = wx + b$$

prediction — slope — intercept



Multiple linear regression



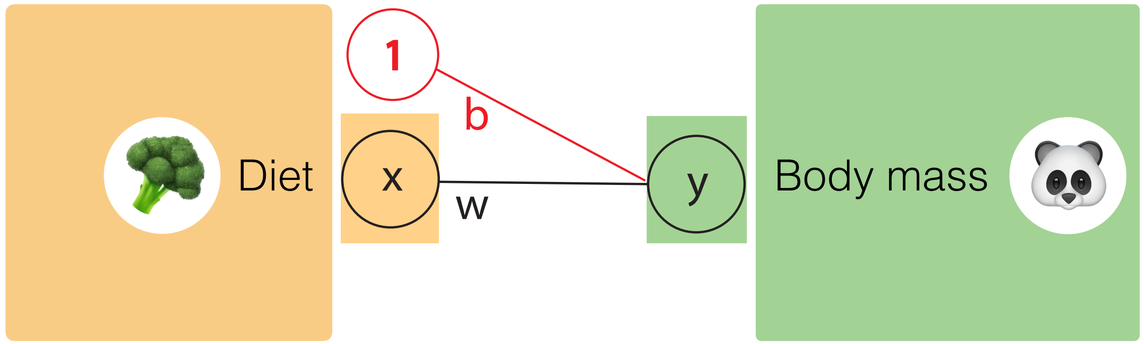
$$y = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b$$

prediction — slopes (effect sizes) — intercept

$$y = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \end{pmatrix} + b$$

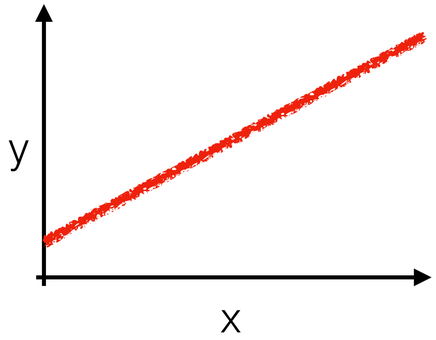
We can re-write it as a matrix multiplication

Linear regression

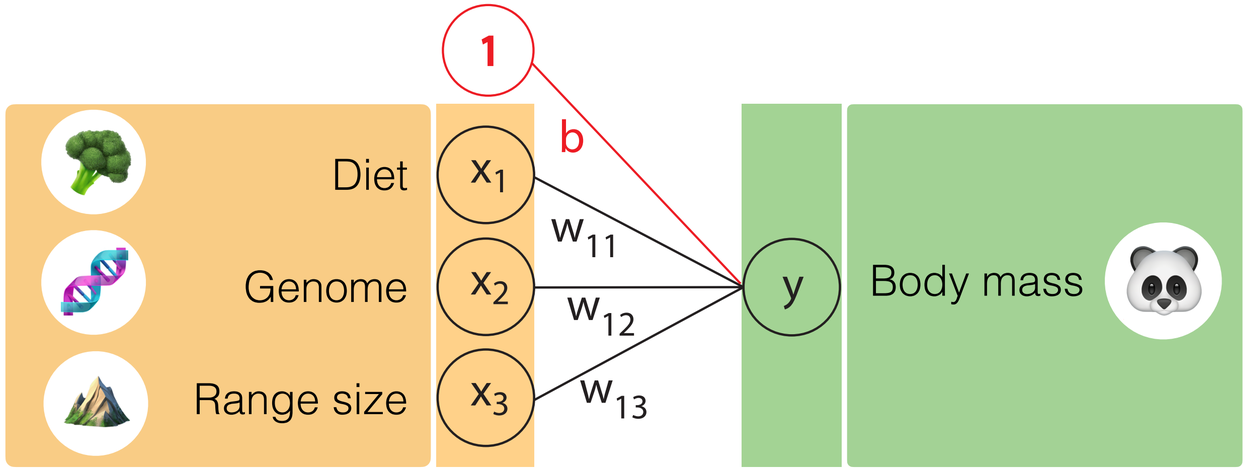


$$y = wx + b$$

prediction — slope — intercept



Multiple linear regression



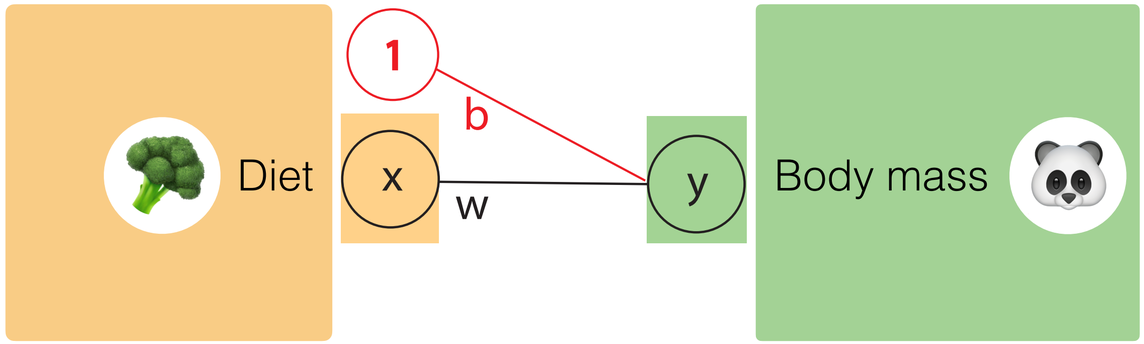
$$y = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b$$

prediction — slopes (effect sizes) — intercept

$$y = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \end{pmatrix} + b$$

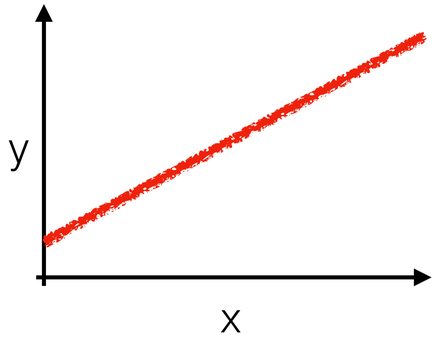
We can re-write it as a matrix multiplication

Linear regression

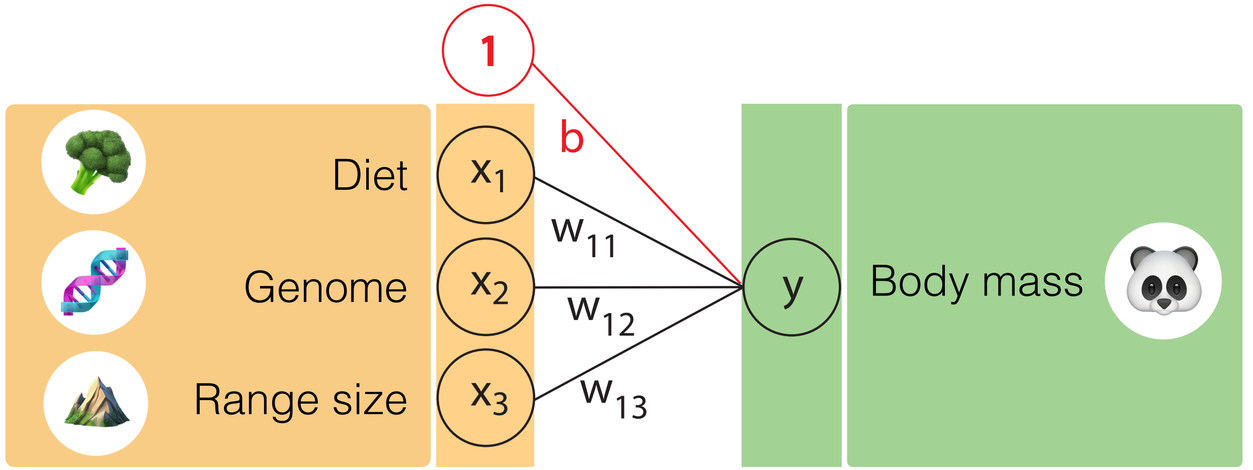


$$y = wx + b$$

prediction — slope — intercept

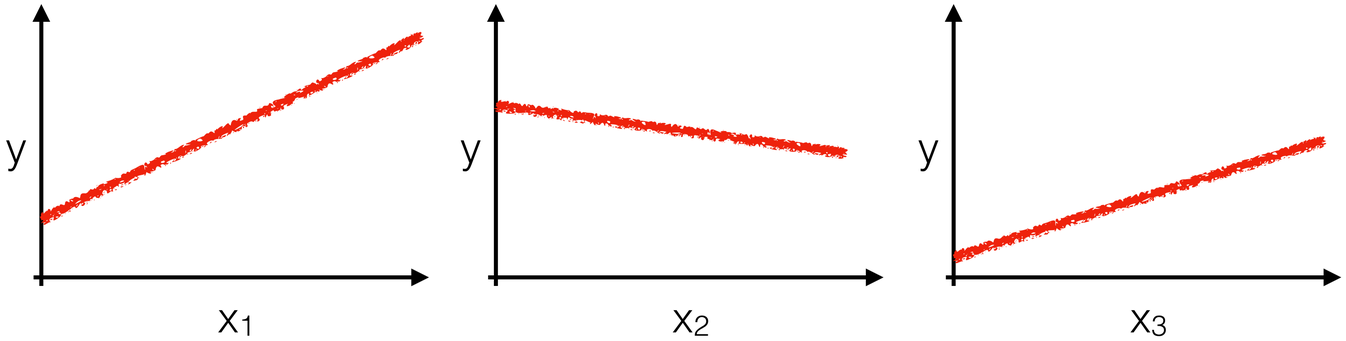


Multiple linear regression

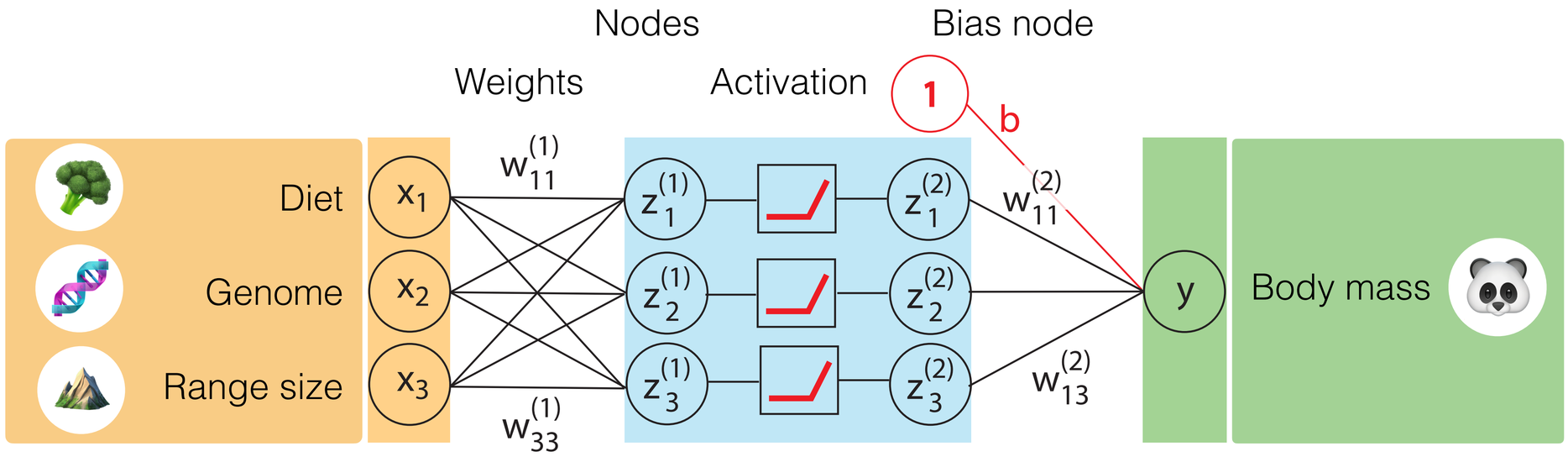


$$y = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b$$

prediction — slopes (effect sizes) — intercept



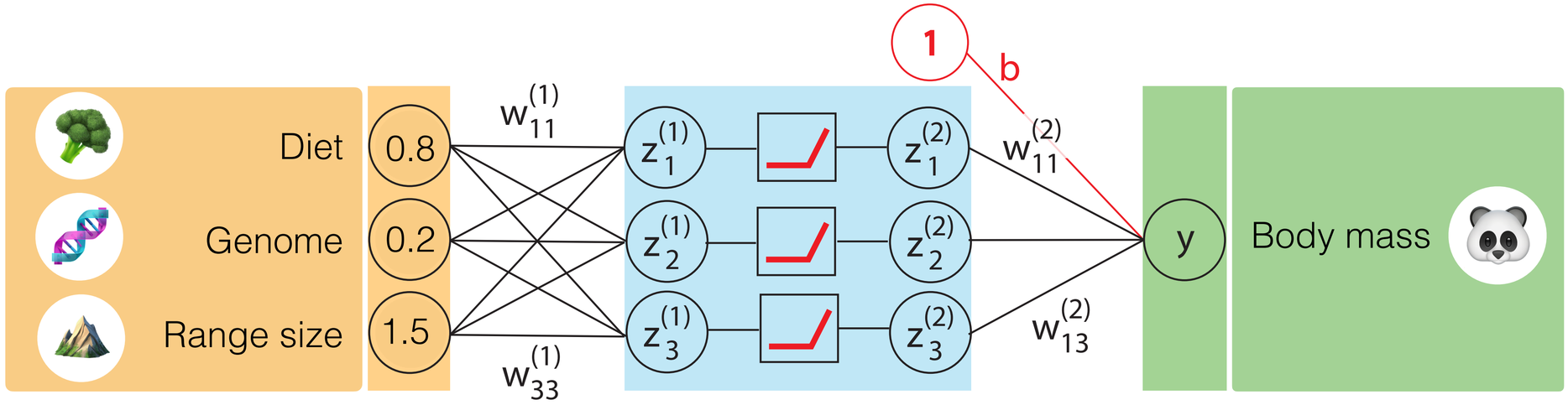
A neural network regression model



$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} = \begin{pmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{pmatrix}$$

$\mathbf{x}$ $\quad$ $\mathbf{w}^{(1)}$ $\quad$ $\mathbf{z}^{(1)}$

A neural network regression model

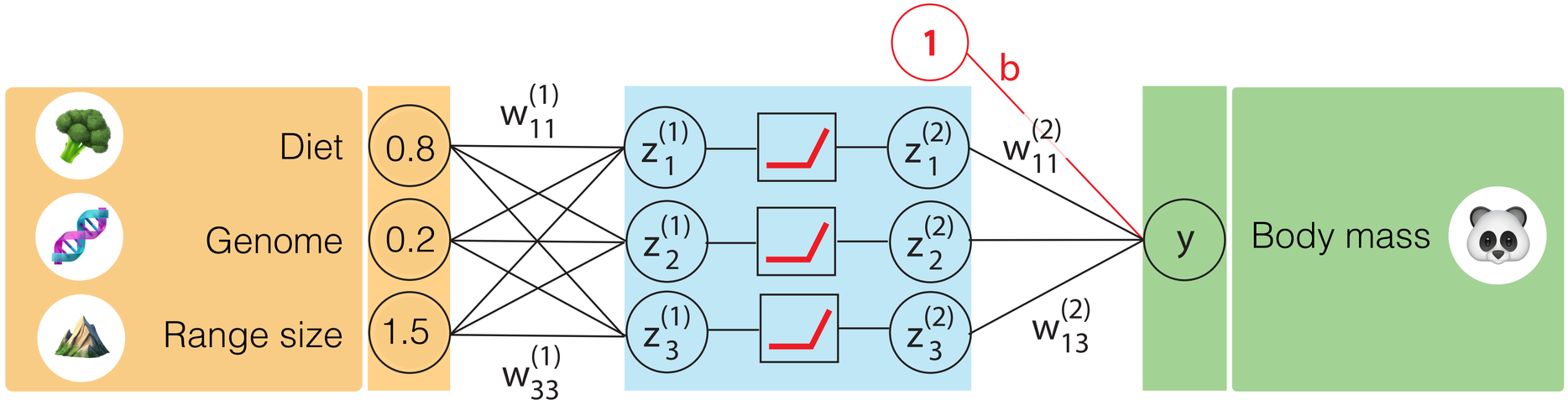


Matrix multiplication
Multiply each **x** by **w** columns

$$\begin{pmatrix} 0.8 \\ 0.2 \\ 1.5 \end{pmatrix} \begin{pmatrix} -0.4 & 1.1 & 1.1 \\ 0.9 & 0.4 & -1.2 \\ 0.6 & 1.3 & 0.2 \end{pmatrix} = \begin{pmatrix} -0.3 & 0.2 & 1.6 \\ 0.7 & 0.1 & -1.8 \\ 0.5 & 0.3 & 0.3 \end{pmatrix}$$

x $\quad$ $w^{(1)}$

A neural network regression model

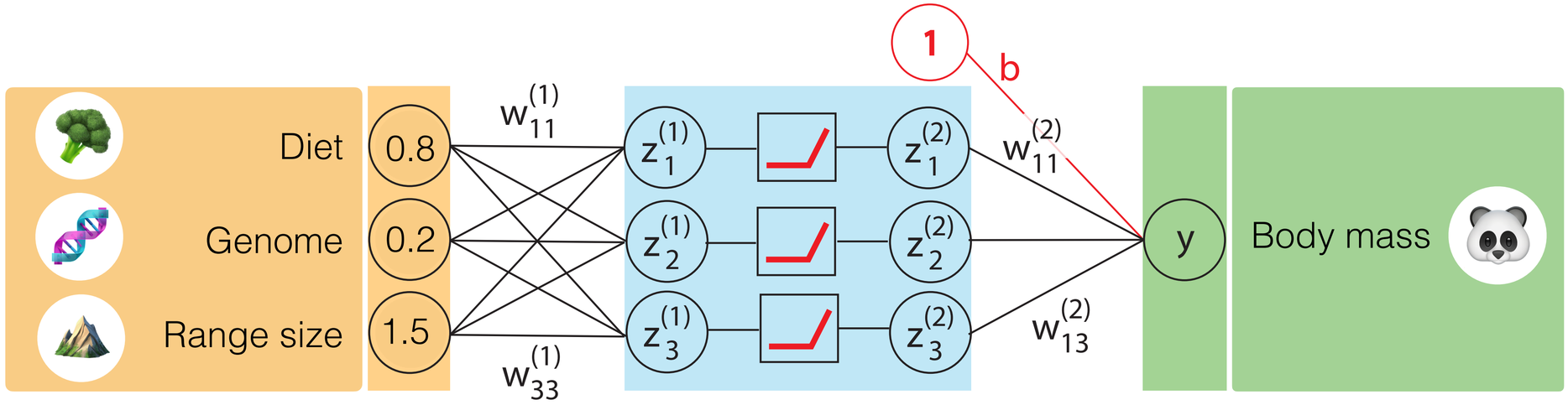


Matrix multiplication
Multiply each **x** by **w** columns

$$\begin{pmatrix} 0.8 \\ 0.2 \\ 1.5 \end{pmatrix} \begin{pmatrix} -0.4 & 1.1 & 1.1 \\ 0.9 & 0.4 & -1.2 \\ 0.6 & 1.3 & 0.2 \end{pmatrix} = \begin{pmatrix} -0.3 & 0.2 & 1.6 \\ 0.7 & 0.1 & -1.8 \\ 0.5 & 0.3 & 0.3 \end{pmatrix}$$

x $\quad$ $w^{(1)}$

A neural network regression model

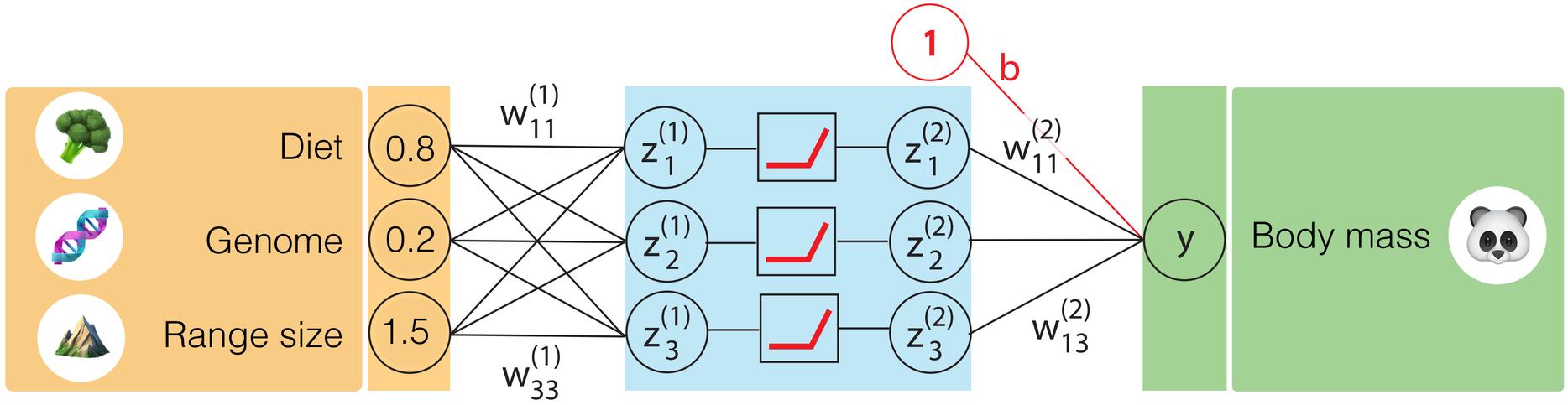


Matrix multiplication
Multiply each **x** by **w** columns

$$\begin{pmatrix} 0.8 \\ 0.2 \\ 1.5 \end{pmatrix} \begin{pmatrix} -0.4 & 1.1 & 1.1 \\ 0.9 & 0.4 & -1.2 \\ 0.6 & 1.3 & 0.2 \end{pmatrix} = \begin{pmatrix} -0.3 & 0.2 & 1.6 \\ 0.7 & 0.1 & -1.8 \\ 0.5 & 0.3 & 0.3 \end{pmatrix}$$

x $\quad$ $w^{(1)}$

A neural network regression model

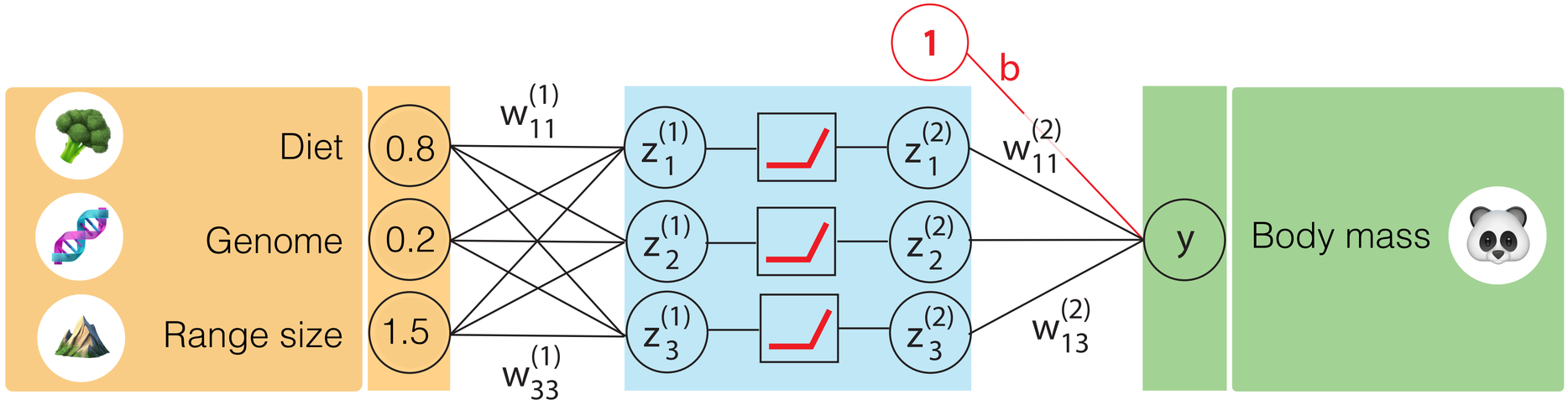


Matrix multiplication
Multiply each **x** by **w** columns and sum by rows.

$$\begin{pmatrix} 0.8 \\ 0.2 \\ 1.5 \end{pmatrix} \begin{pmatrix} -0.4 & 1.1 & 1.1 \\ 0.9 & 0.4 & -1.2 \\ 0.6 & 1.3 & 0.2 \end{pmatrix} = \begin{pmatrix} -0.3 & 0.2 & 1.6 \\ 0.7 & 0.1 & -1.8 \\ 0.5 & 0.3 & 0.3 \end{pmatrix} = \begin{pmatrix} 1.5 \\ -1.0 \\ 1.1 \end{pmatrix}$$

$\mathbf{x}$ $\quad$ $\mathbf{w}^{(1)}$ $\quad$ $\mathbf{z}^{(1)}$

A neural network regression model

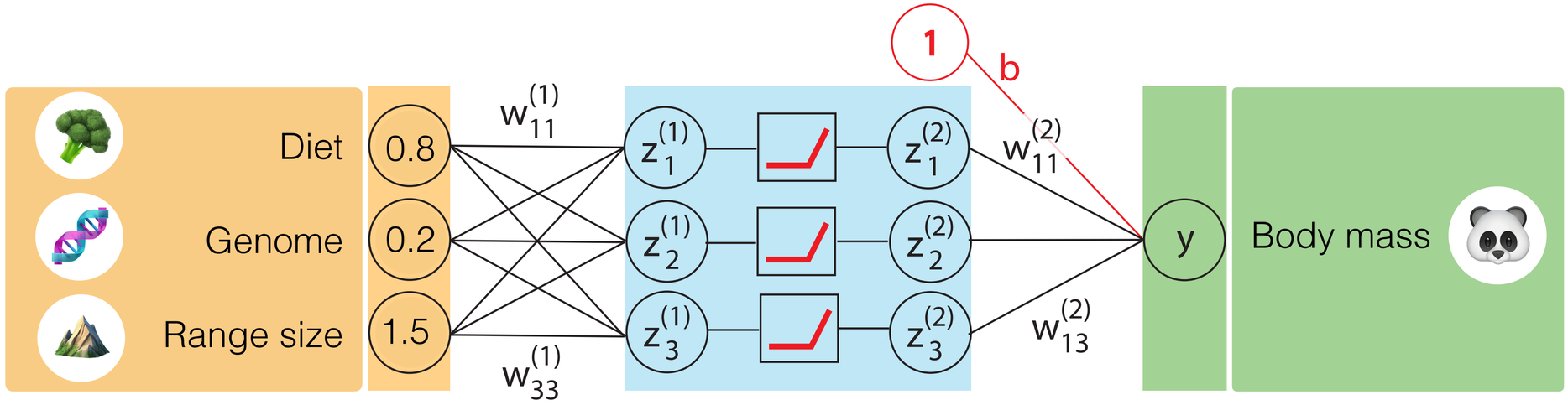


Matrix multiplication
Multiply each **x** by **w** columns and sum by rows.

$$\begin{pmatrix} 0.8 \\ 0.2 \\ 1.5 \end{pmatrix} \begin{pmatrix} -0.4 & 1.1 & 1.1 \\ 0.9 & 0.4 & -1.2 \\ 0.6 & 1.3 & 0.2 \end{pmatrix} = \begin{pmatrix} -0.3 & 0.2 & 1.6 \\ 0.7 & 0.1 & -1.8 \\ 0.5 & 0.3 & 0.3 \end{pmatrix} = \begin{pmatrix} 1.5 \\ -1.0 \\ 1.1 \end{pmatrix}$$

$\mathbf{x}$ $\quad$ $\mathbf{w}^{(1)}$ $\quad$ $\mathbf{z}^{(1)}$

A neural network regression model

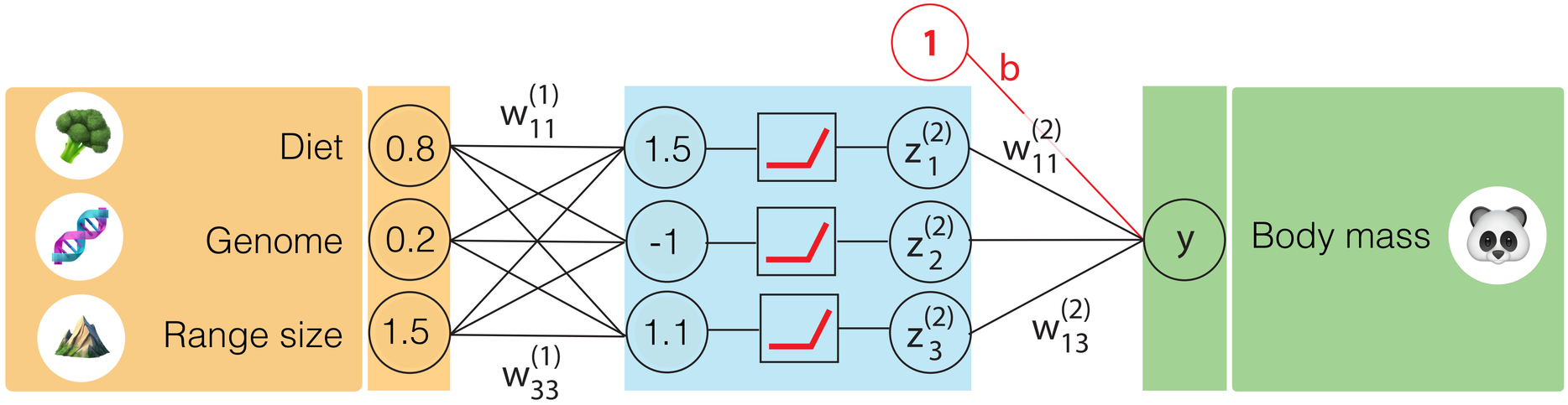


Matrix multiplication
Multiply each **x** by **w** columns and sum by rows.

$$\begin{pmatrix} 0.8 \\ 0.2 \\ 1.5 \end{pmatrix} \begin{pmatrix} -0.4 & 1.1 & 1.1 \\ 0.9 & 0.4 & -1.2 \\ 0.6 & 1.3 & 0.2 \end{pmatrix} = \begin{pmatrix} -0.3 & 0.2 & 1.6 \\ 0.7 & 0.1 & -1.8 \\ 0.5 & 0.3 & 0.3 \end{pmatrix} = \begin{pmatrix} 1.5 \\ -1.0 \\ 1.1 \end{pmatrix}$$

$\mathbf{x}$ $\quad$ $\mathbf{w}^{(1)}$ $\quad$ $\mathbf{z}^{(1)}$

A neural network regression model



Matrix multiplication
Multiply each **x** by **w** columns and sum by rows.

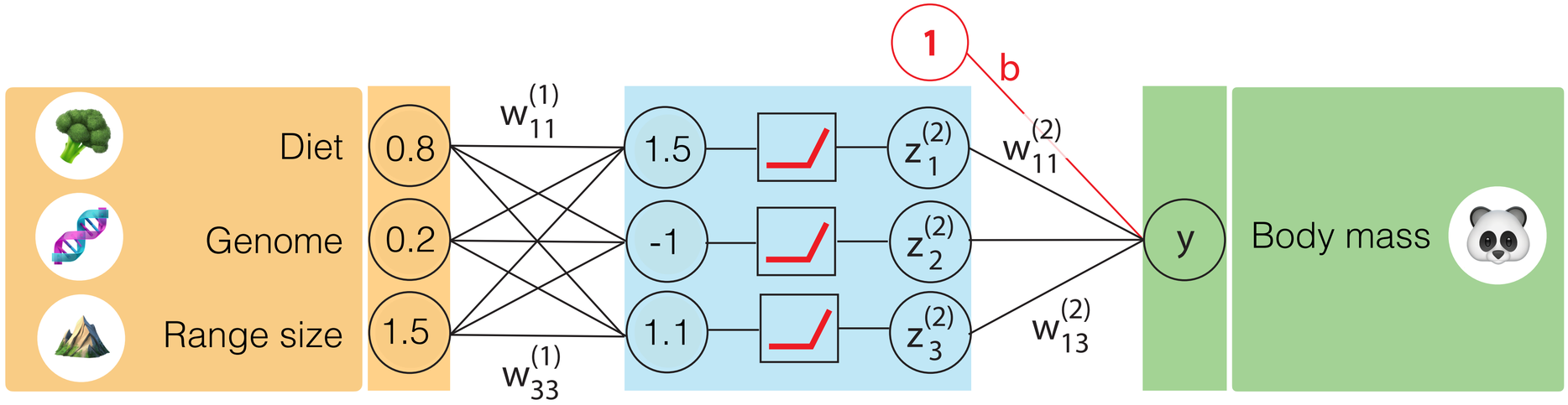
$$\begin{pmatrix} 0.8 \\ 0.2 \\ 1.5 \end{pmatrix} \begin{pmatrix} -0.4 & 1.1 & 1.1 \\ 0.9 & 0.4 & -1.2 \\ 0.6 & 1.3 & 0.2 \end{pmatrix} = \begin{pmatrix} -0.3 & 0.2 & 1.6 \\ 0.7 & 0.1 & -1.8 \\ 0.5 & 0.3 & 0.3 \end{pmatrix} = \begin{pmatrix} 1.5 \\ -1.0 \\ 1.1 \end{pmatrix}$$

$\mathbf{x}$ $\quad$ $\mathbf{w}^{(1)}$ $\quad$ $\mathbf{z}^{(1)}$

This is comfortably computed in 1 line in Python:

```
z1 = np.einsum('j,ij->i', x, w1)
```

A neural network regression model

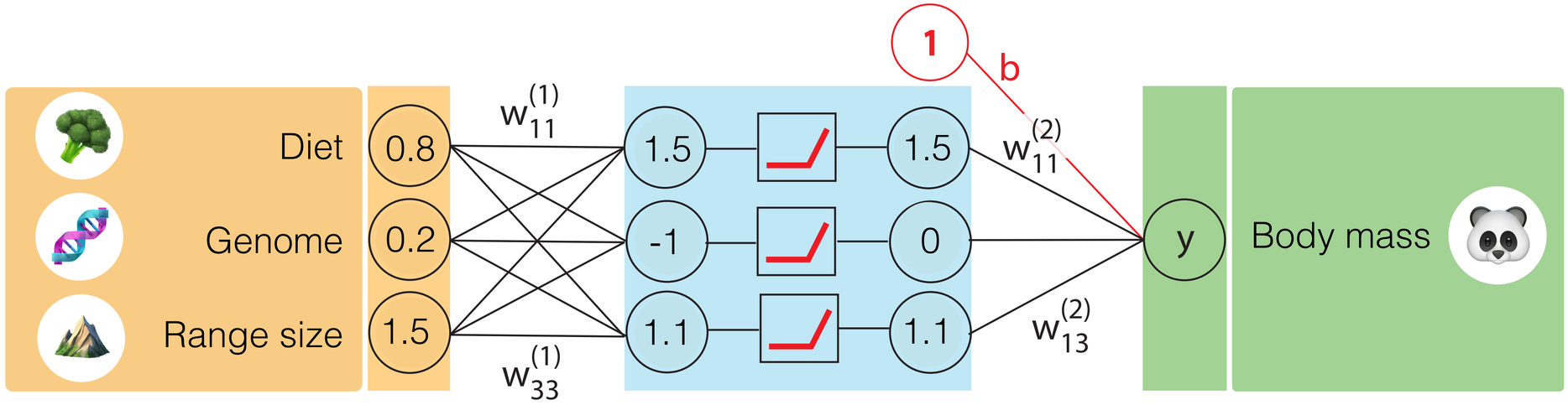


$$\begin{pmatrix} 0.8 \\ 0.2 \\ 1.5 \end{pmatrix} \begin{pmatrix} -0.4 & 1.1 & 1.1 \\ 0.9 & 0.4 & -1.2 \\ 0.6 & 1.3 & 0.2 \end{pmatrix} = \begin{pmatrix} -0.3 & 0.2 & 1.6 \\ 0.7 & 0.1 & -1.8 \\ 0.5 & 0.3 & 0.3 \end{pmatrix} = \begin{pmatrix} 1.5 \\ -1.0 \\ 1.1 \end{pmatrix}$$

$\mathbf{x}$ $\quad$ $\mathbf{w}^{(1)}$ $\quad$ $\mathbf{z}^{(1)}$

Each item in vector $\mathbf{z}^{(1)}$ is now a function of all input features

A neural network regression model



$$\begin{pmatrix} 1.5 \\ -1.0 \\ 1.1 \end{pmatrix} \quad = \quad \begin{pmatrix} 1.5 \\ 0 \\ 1.1 \end{pmatrix}$$

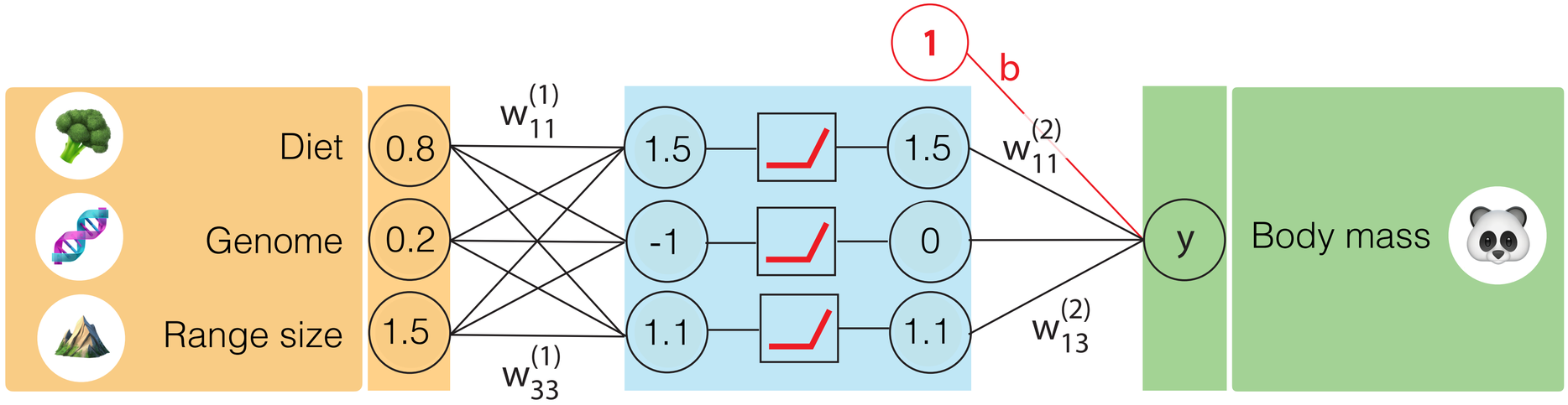
$z^{(1)}$ $z^{(2)}$

ReLU function
non-linear activation function:
negative values are (arbitrarily) set to 0

In Python:

```
z2 = z1 * (z1 > 0)
```

A neural network regression model

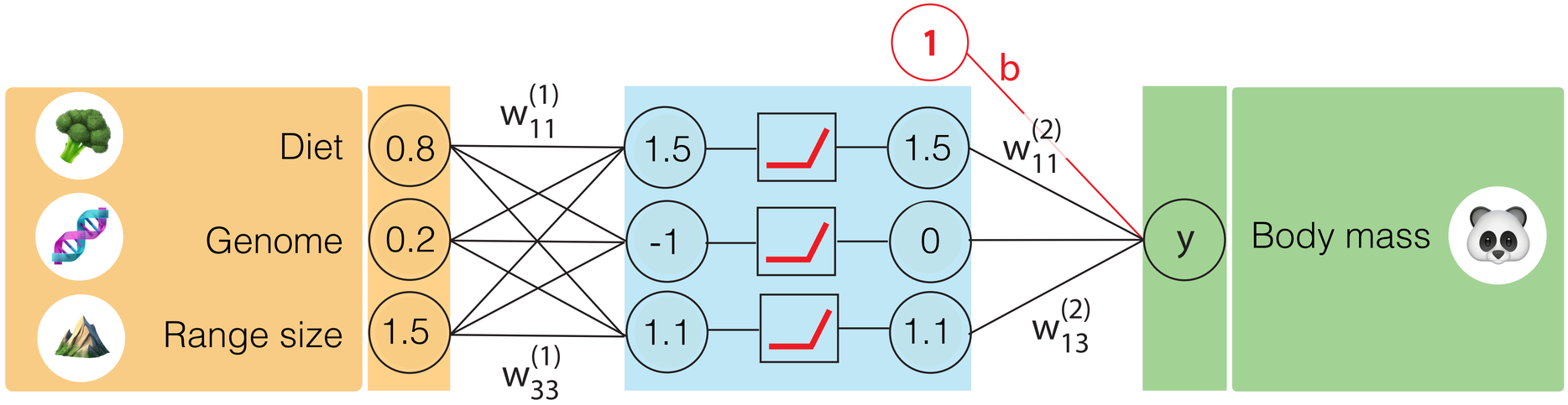


Bias node (cf. intercept)

$$\begin{pmatrix} 1 \\ 1.5 \\ 0 \\ 1.1 \end{pmatrix} \begin{pmatrix} -0.4 & 0.1 & 2.0 & 2.6 \end{pmatrix} = \begin{pmatrix} -0.4 & 0.2 & 0 & 2.9 \end{pmatrix}$$

$z^{(2)}$ $w^{(1)}$

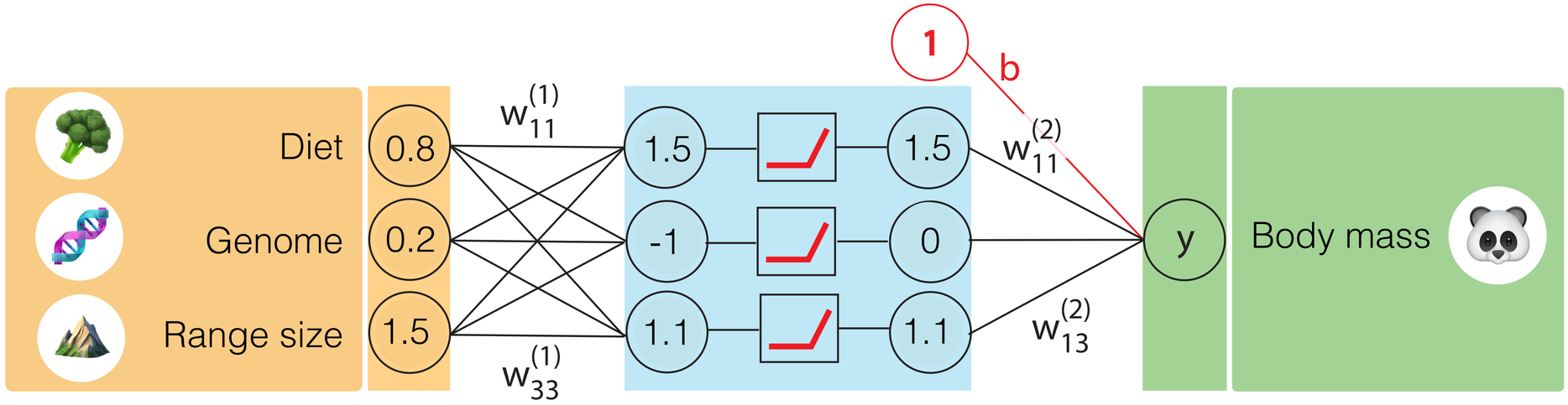
A neural network regression model



$$\begin{pmatrix} 1 \\ 1.5 \\ 0 \\ 1.1 \end{pmatrix} \begin{pmatrix} -0.4 & 0.1 & 2.0 & 2.6 \end{pmatrix} = \begin{pmatrix} -0.4 & 0.2 & 0 & 2.9 \end{pmatrix}$$

$z^{(2)}$ $w^{(1)}$

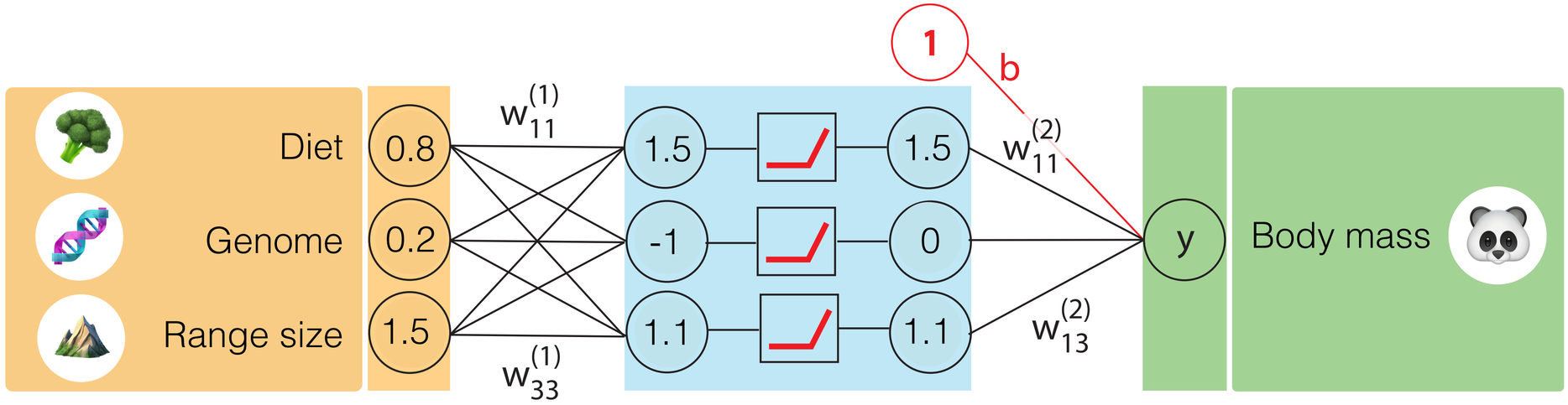
A neural network regression model



$$\begin{pmatrix} 1 \\ 1.5 \\ 0 \\ 1.1 \end{pmatrix} \begin{pmatrix} -0.4 & 0.1 & 2.0 & 2.6 \end{pmatrix} = \begin{pmatrix} -0.4 & 0.2 & 0 & 2.9 \end{pmatrix}$$

$z^{(2)}$ $w^{(1)}$

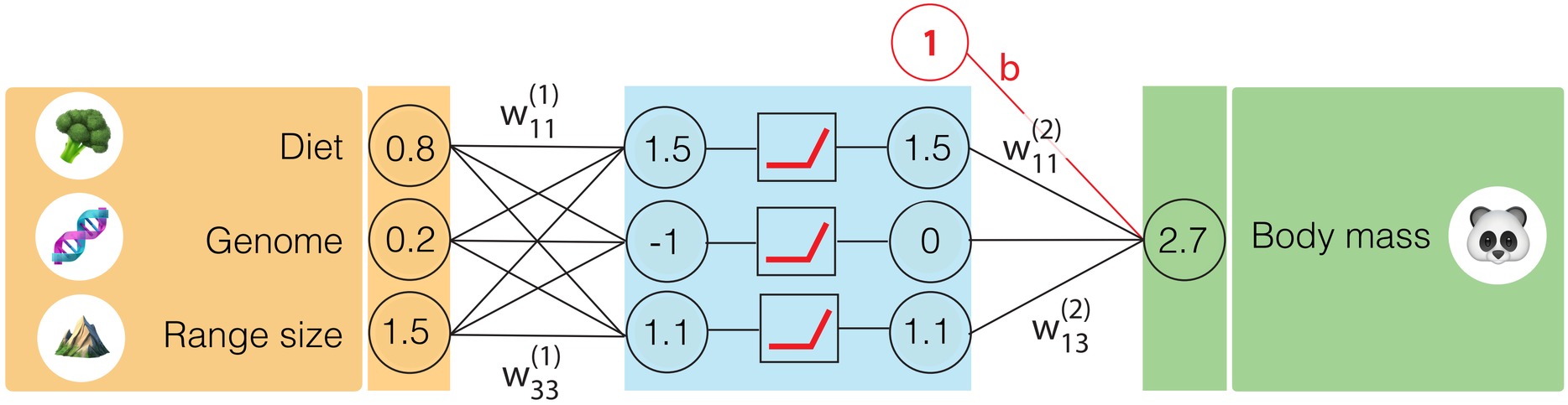
A neural network regression model



$$\begin{pmatrix} 1 \\ 1.5 \\ 0 \\ 1.1 \end{pmatrix} \begin{pmatrix} -0.4 & 0.1 & 2.0 & 2.6 \end{pmatrix} = \begin{pmatrix} -0.4 & 0.2 & 0 & 2.9 \end{pmatrix}$$

$z^{(2)}$ $w^{(1)}$

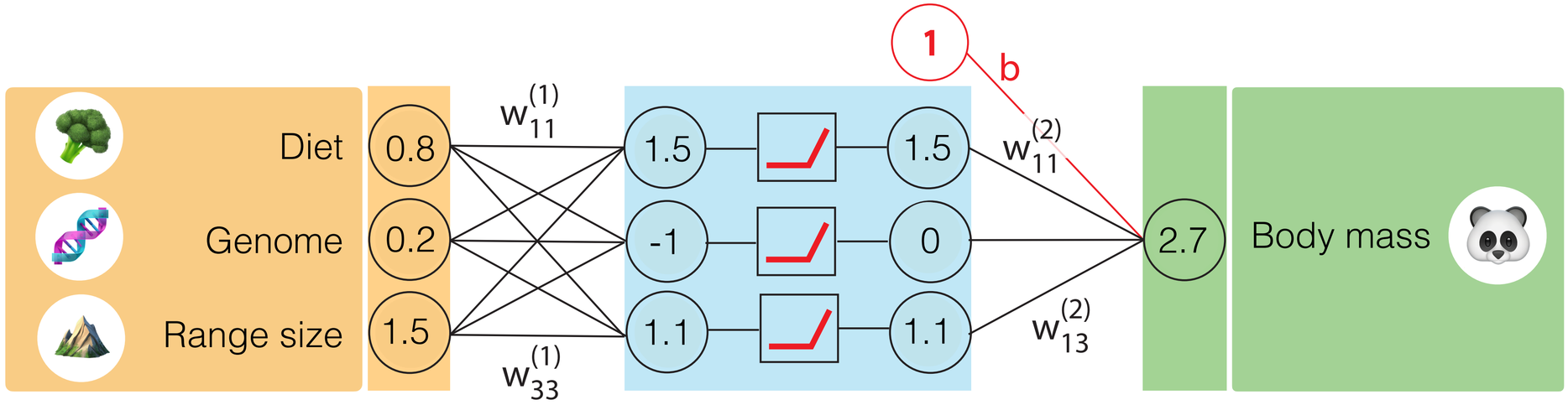
A neural network regression model



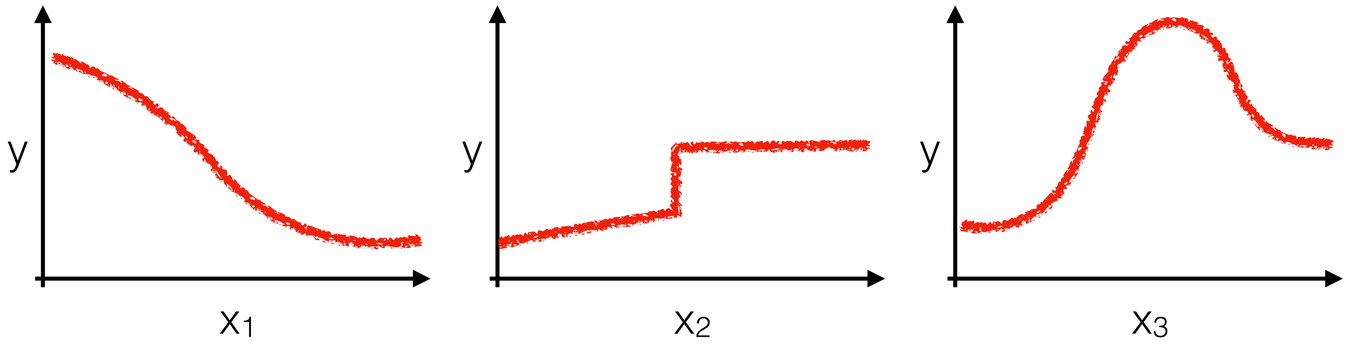
$$\begin{pmatrix} 1 \\ 1.5 \\ 0 \\ 1.1 \end{pmatrix} \begin{pmatrix} -0.4 & 0.1 & 2.0 & 2.6 \end{pmatrix} = \begin{pmatrix} -0.4 & 0.2 & 0 & 2.9 \end{pmatrix} = 2.7$$

$z^{(2)}$ $w^{(1)}$ y

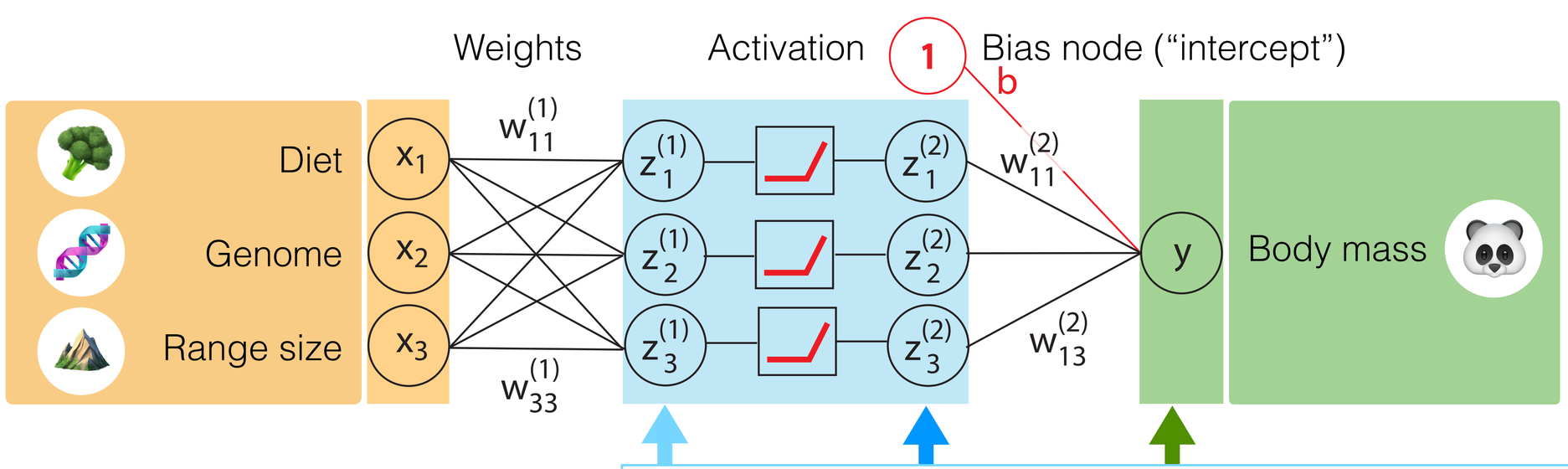
A neural network regression model

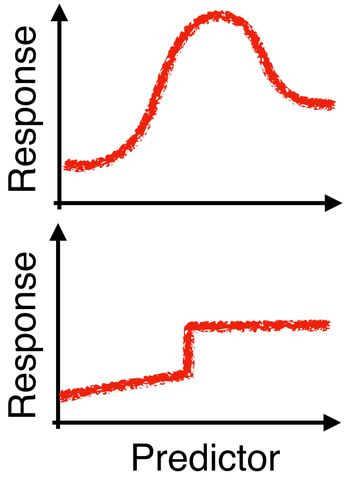


A neural network can approximate virtually any function



Neural network





$$z_i^{(1)} = w_{i1}^{(1)} x_1 + w_{i2}^{(1)} x_2 + w_{i3}^{(1)} x_3$$

Intermediate (hidden) values $z^{(1)}$: linear function of all predictors

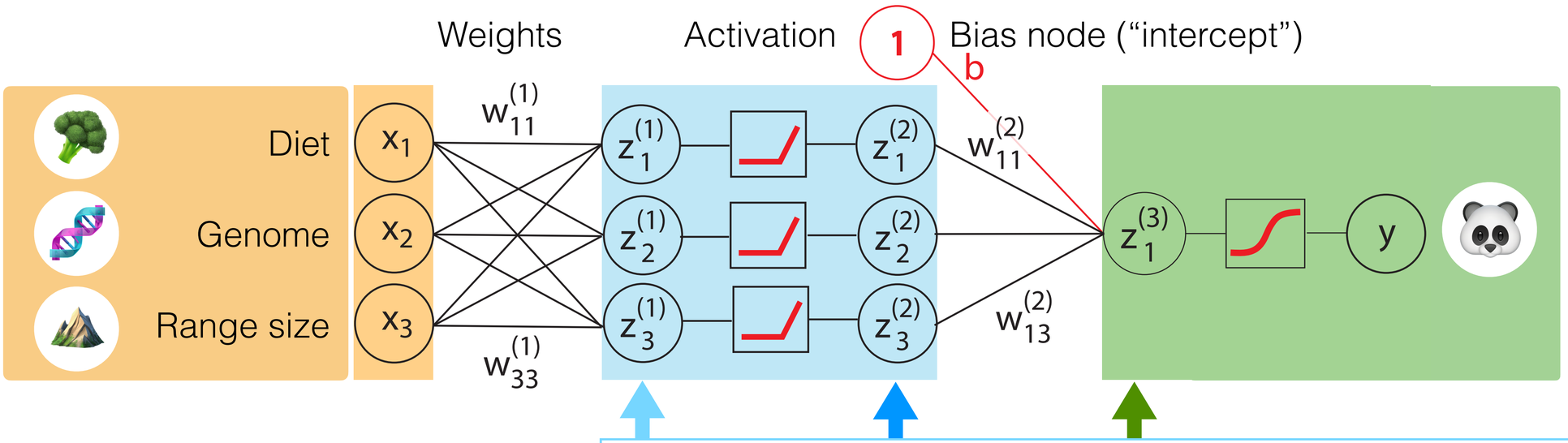
$$z_i^{(2)} = \max(0, z_i^{(1)})$$

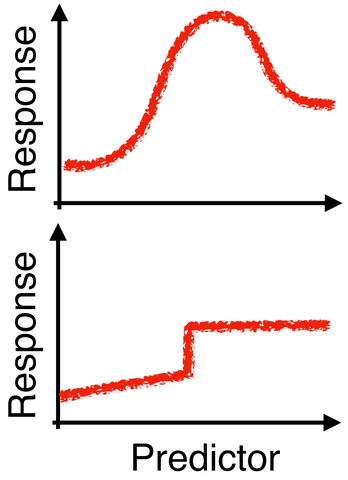
Intermediate (hidden) values $z^{(2)}$: non-linear function of $z^{(1)}$

$$y = w_{i1}^{(2)} z_1^{(2)} + w_{i2}^{(2)} z_2^{(2)} + w_{i3}^{(2)} z_3^{(2)}$$

Output: linear function of $z^{(2)}$

Neural network





$$z_i^{(1)} = w_{i1}^{(1)} x_1 + w_{i2}^{(1)} x_2 + w_{i3}^{(1)} x_3$$

Intermediate (hidden) values $z^{(1)}$: linear function of all predictors

$$z_i^{(2)} = \max(0, z_i^{(1)})$$

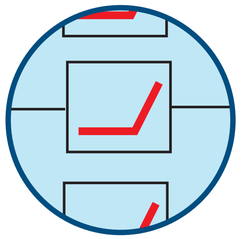
Intermediate (hidden) values $z^{(2)}$: non-linear function of $z^{(1)}$

$$y = w_{i1}^{(2)} z_1^{(2)} + w_{i2}^{(2)} z_2^{(2)} + w_{i3}^{(2)} z_3^{(2)}$$

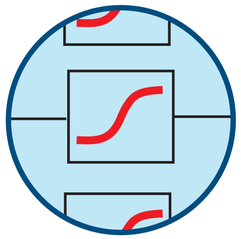
Output: non-linear function of $z^{(2)}$

Parameterization of a neural network

Activation functions



ReLU: rectified linear unit

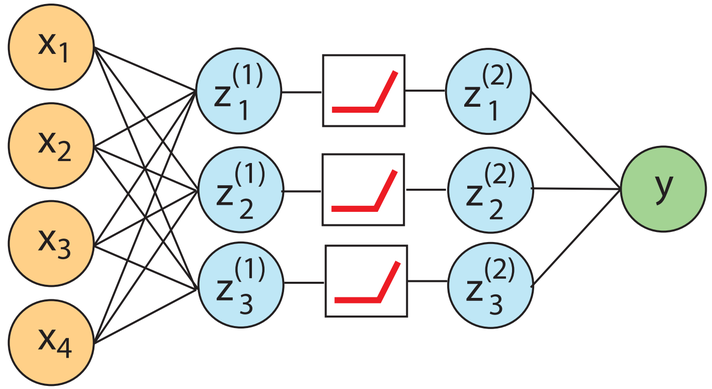


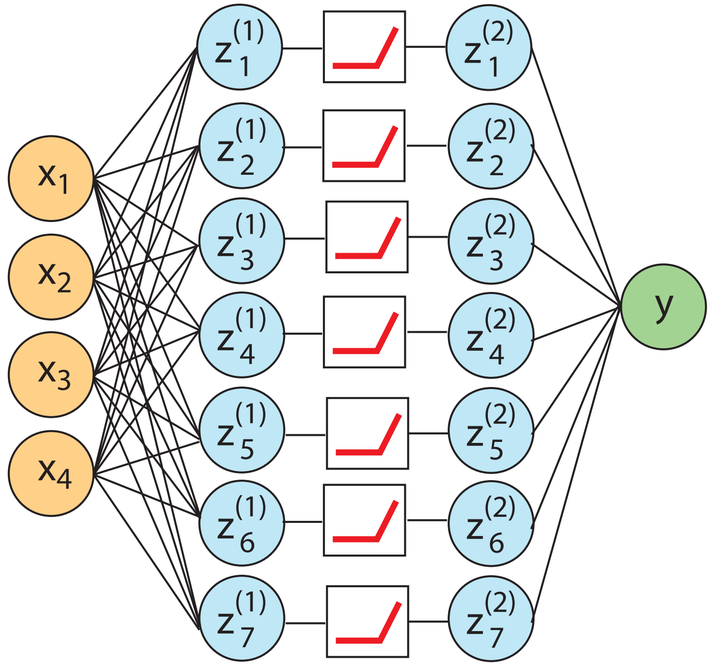
Sigmoid

```
ReLU(x) = max(0, x)
```

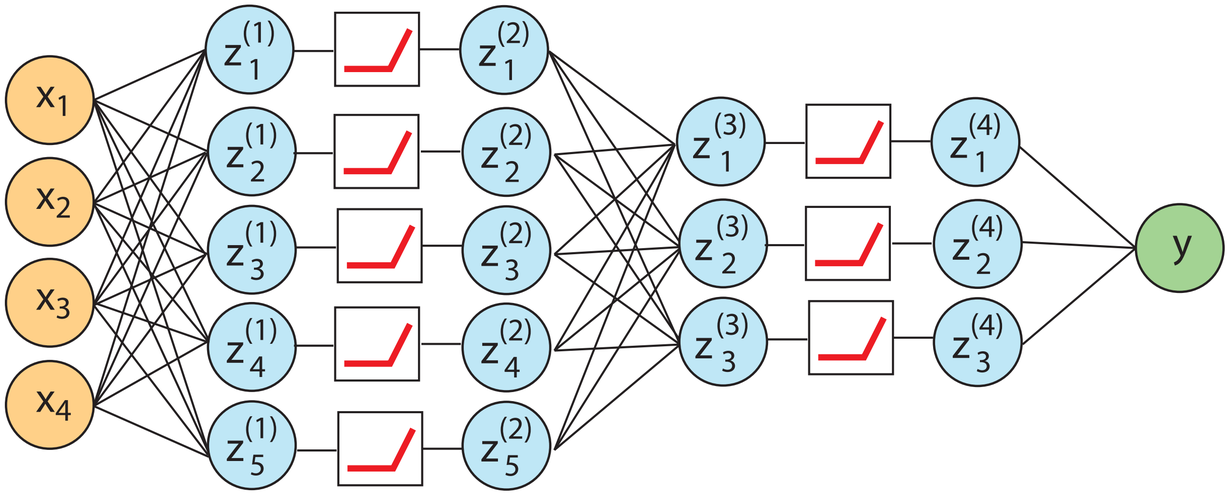
```
sigmoid(x) = 1 / (1 + exp(-x))
```

Number of nodes

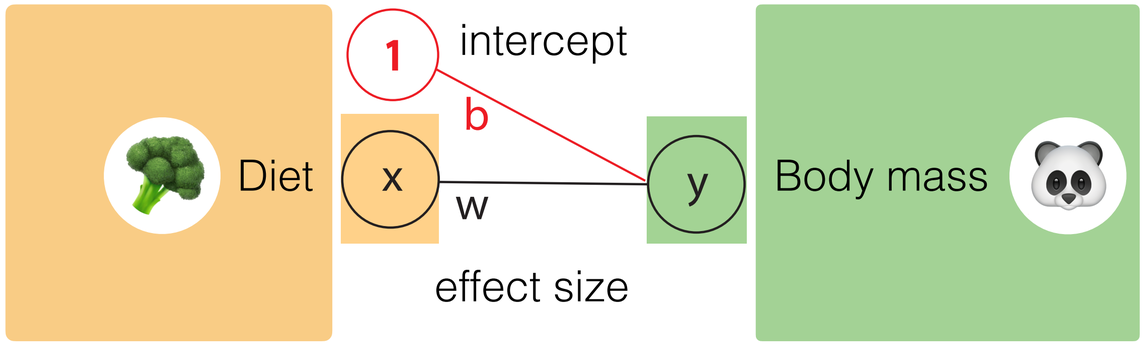




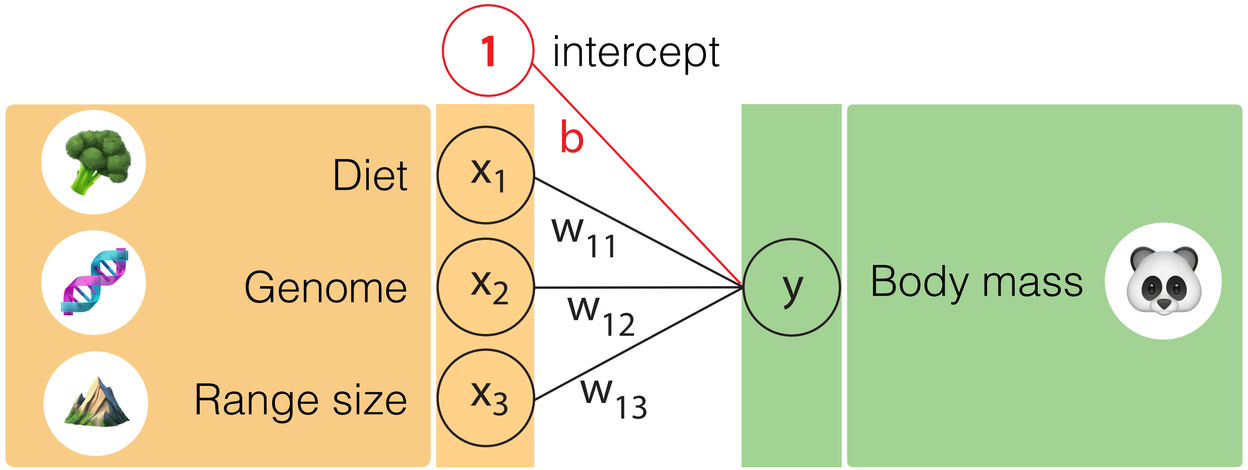
Number of hidden layers (deep NNs)



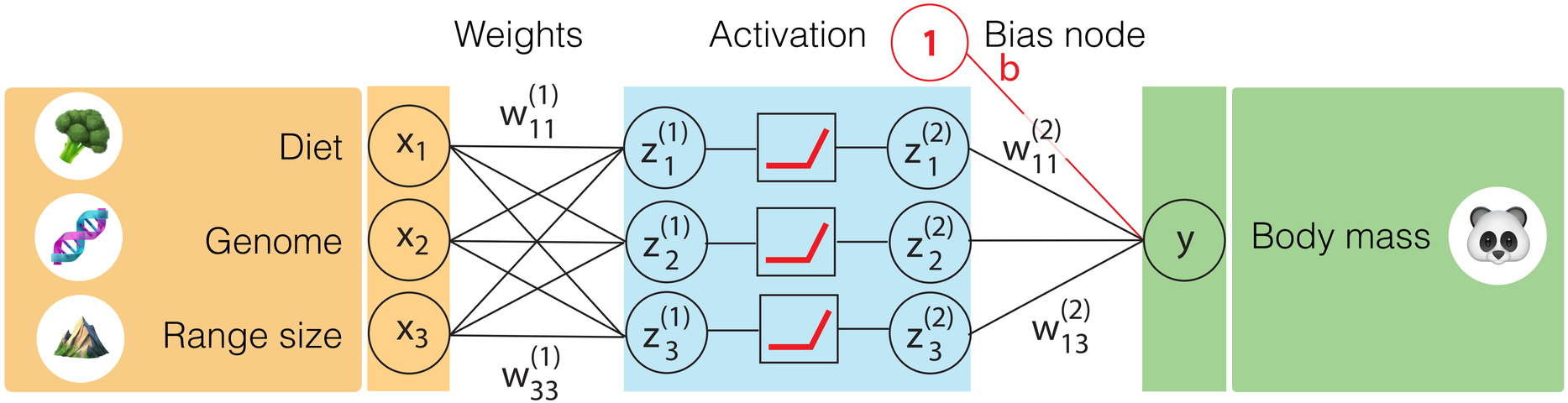
Linear regression



Multiple linear regression



Neural network

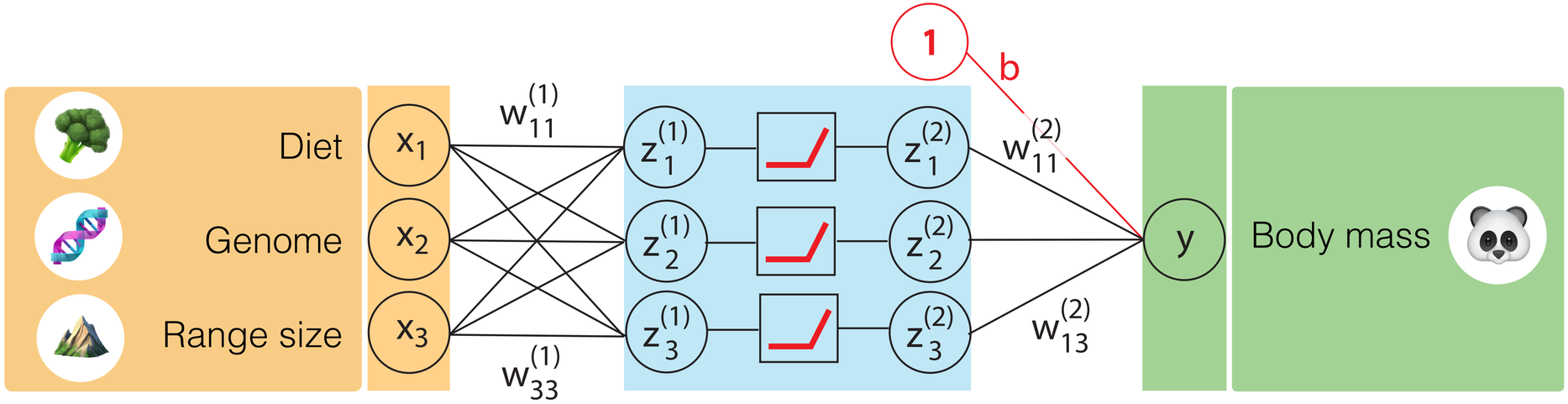


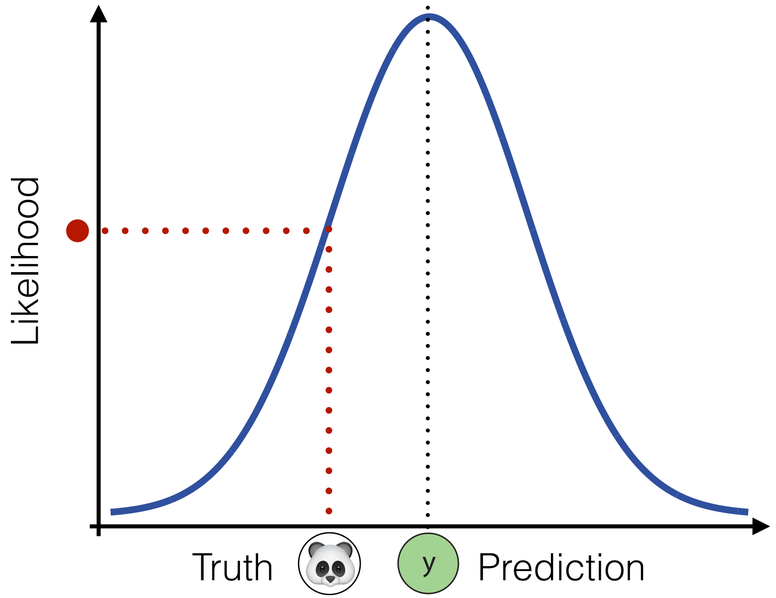
Pros and cons

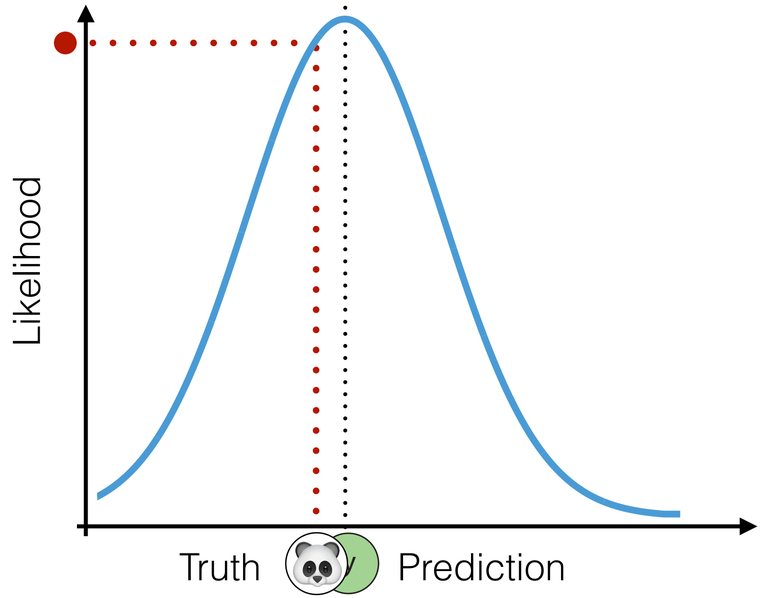
- Non-linear responses
- Interaction among predictors

- No interpretable effect sizes
- Over-parameterized

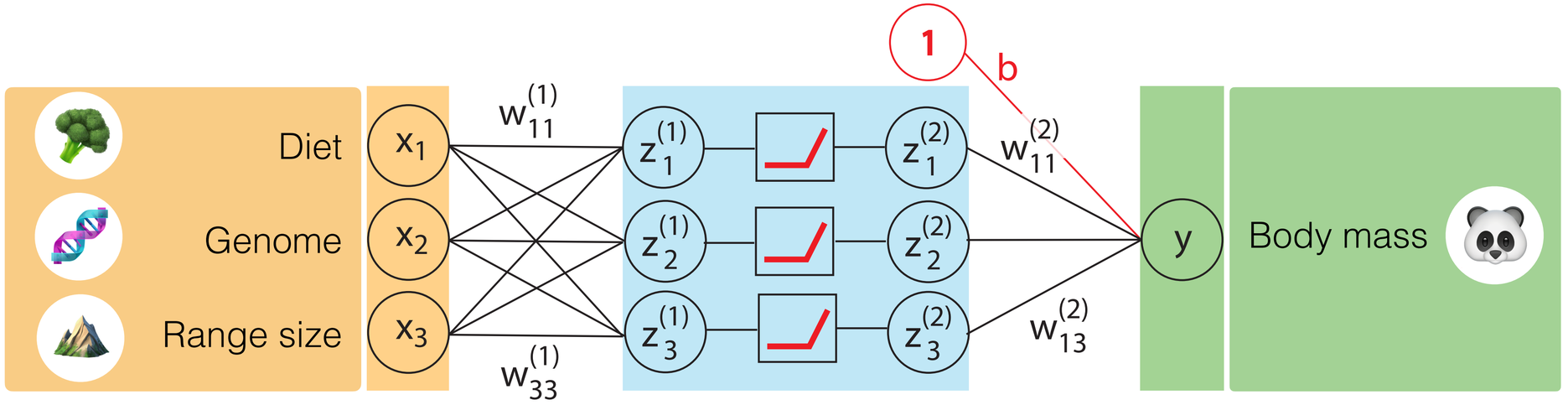
Training a neural network: updating the parameter values to improve the predictions





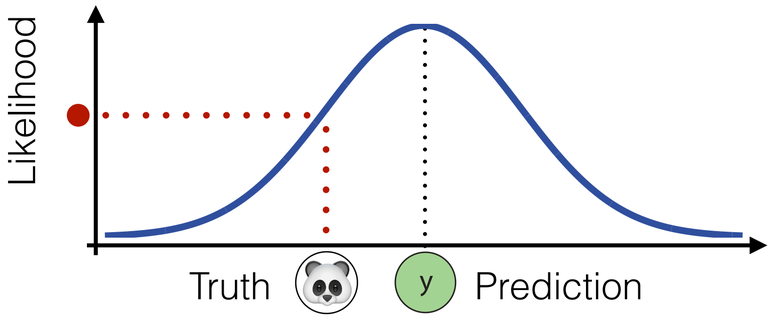


Training a BNN



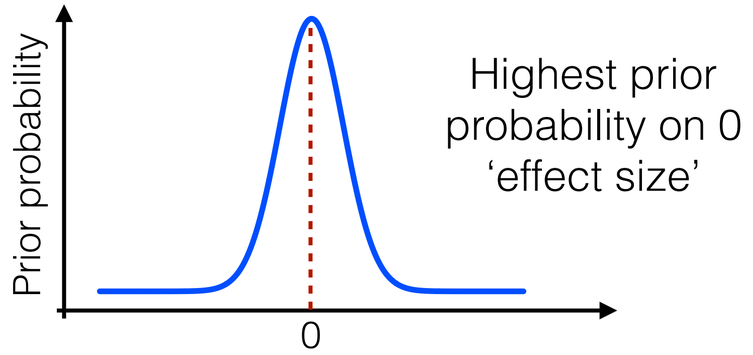
Likelihood function: the normal density function

$$P(\text{🐼}|\text{y},\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\text{🐼}-\text{y})^2}{2\sigma^2}}$$



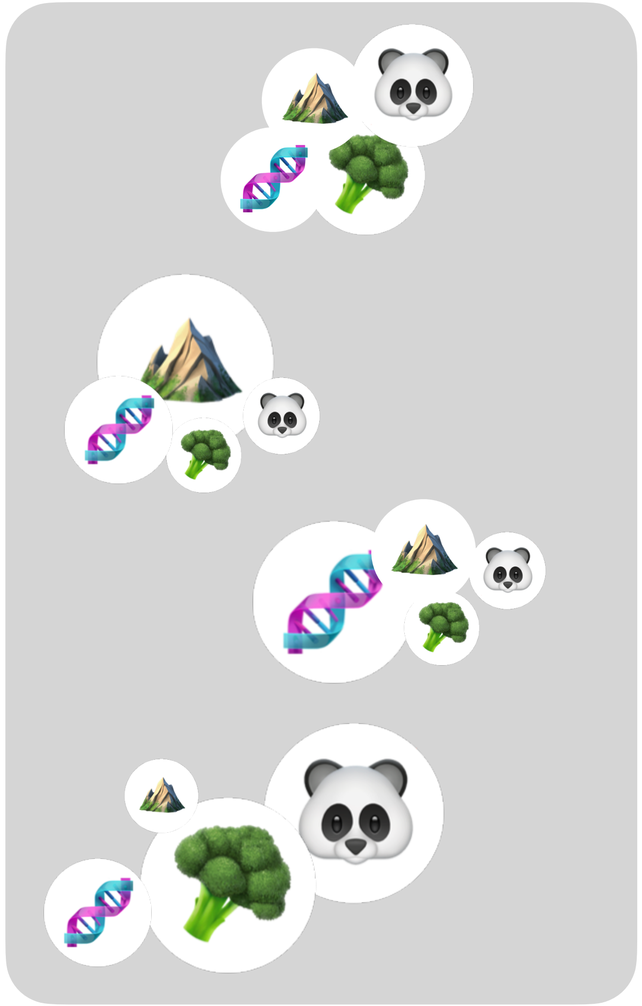
Priors on the weights

$$P(w) \sim \mathcal{N}(0,1)$$

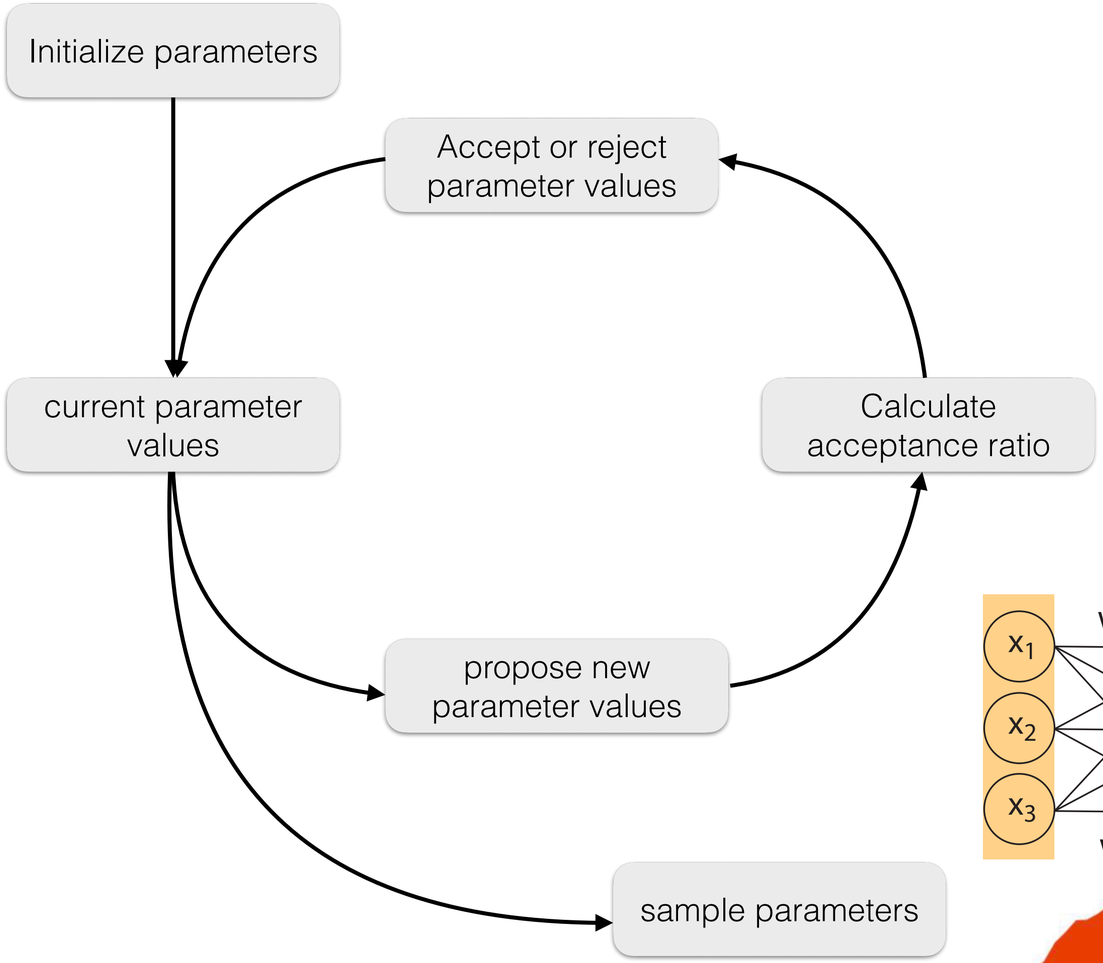


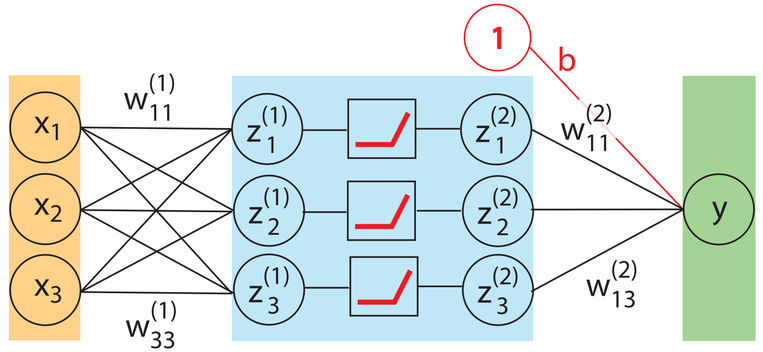
Training a BNN

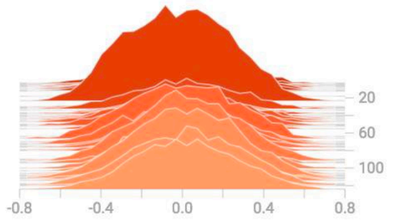
Data (‘training set’)

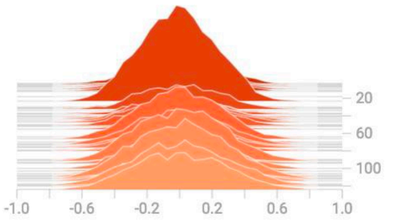


Posterior sampling algorithm, e.g. MCMC or HMC



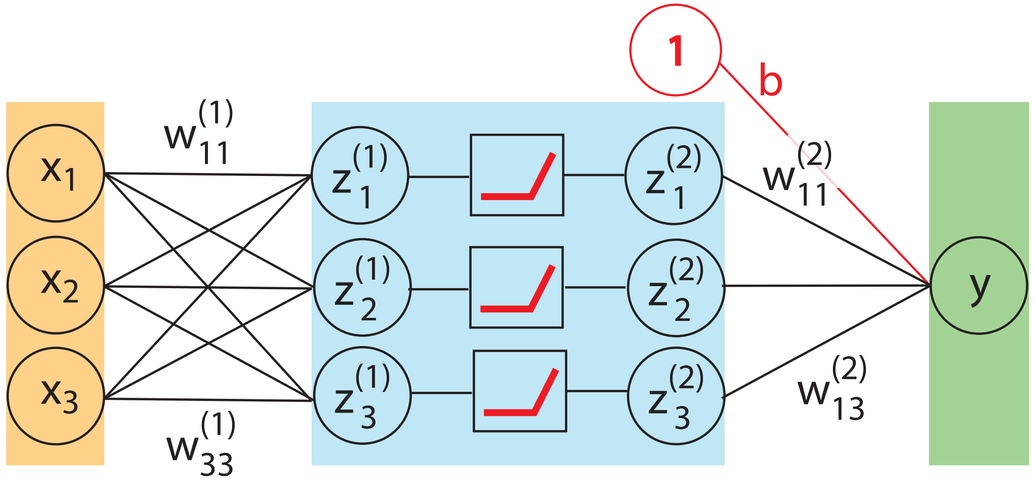




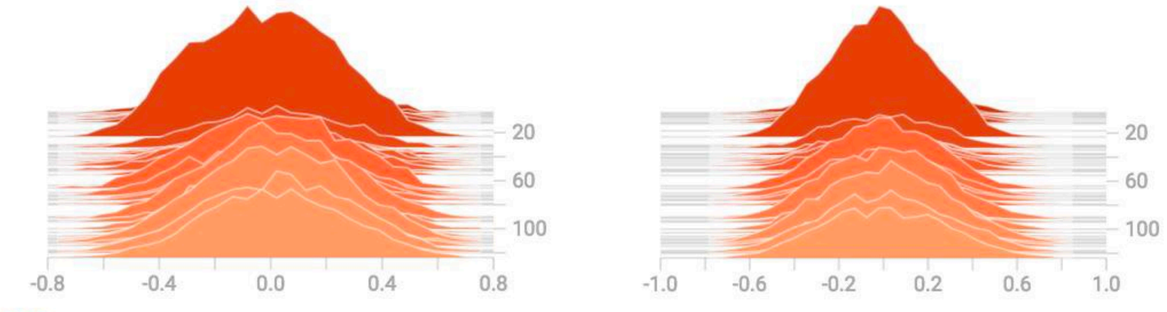


Predictions using a trained BNN

Predicting body mass from
diet, genome, range size

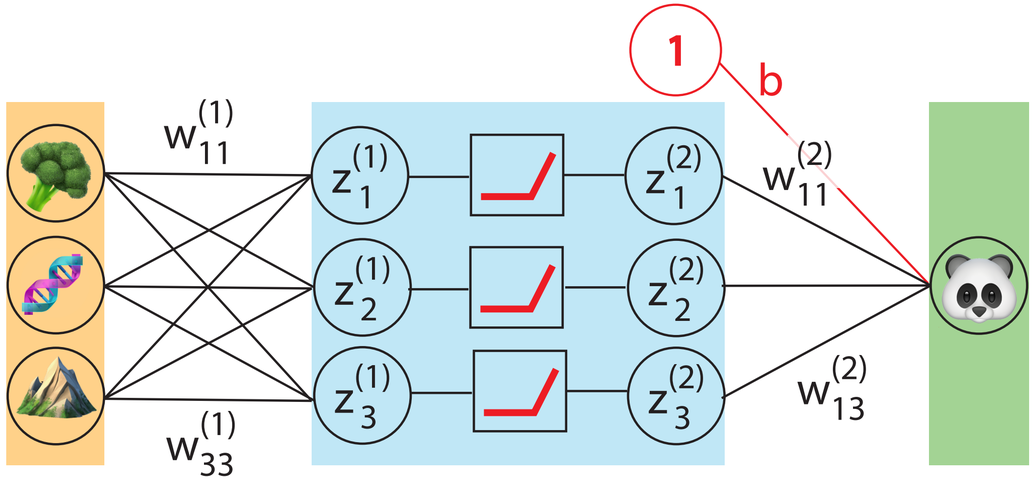


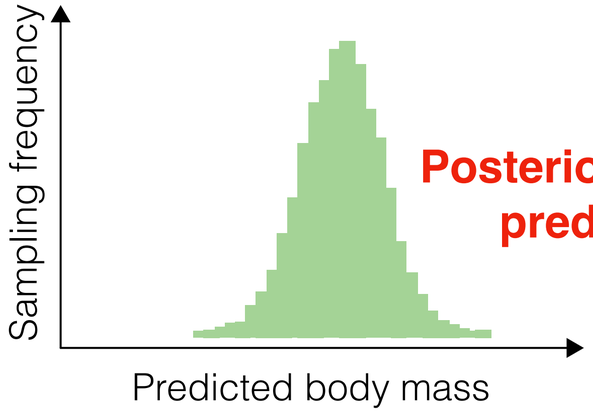
Posterior samples
of the weights



Predictions using a trained BNN

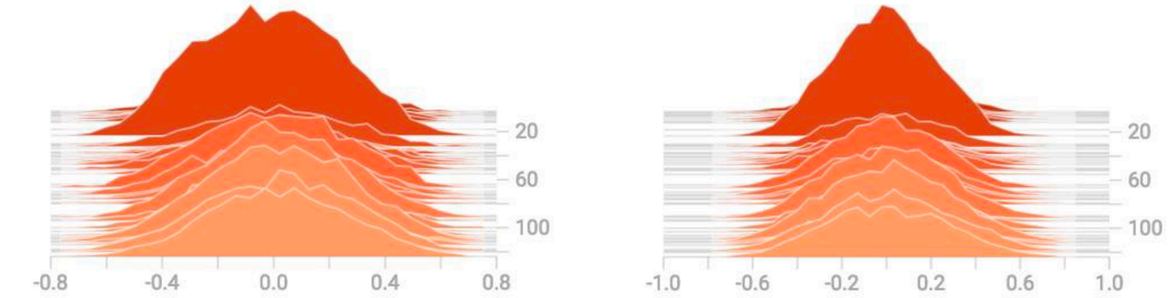
Predicting body mass from diet, genome, range size



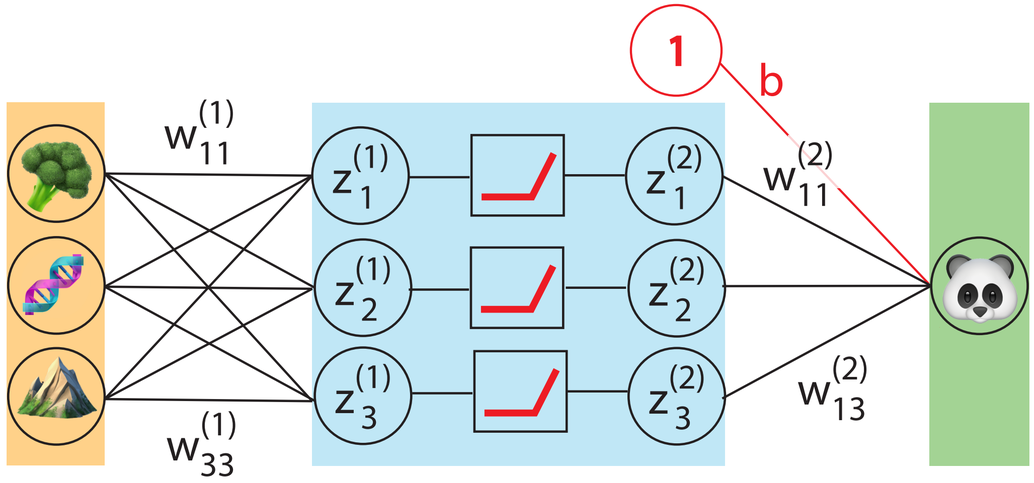


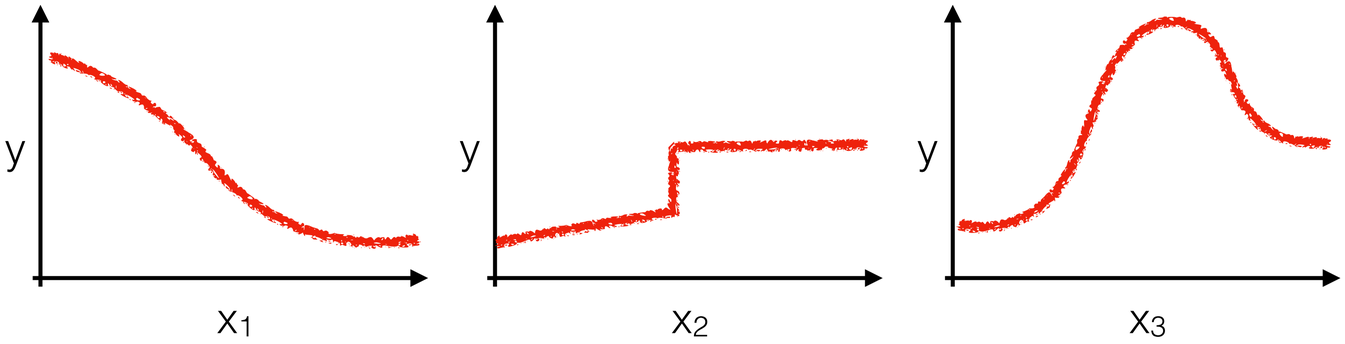
Posterior distribution of predicted values

Posterior samples of the weights



Neural networks can approximate virtually any function



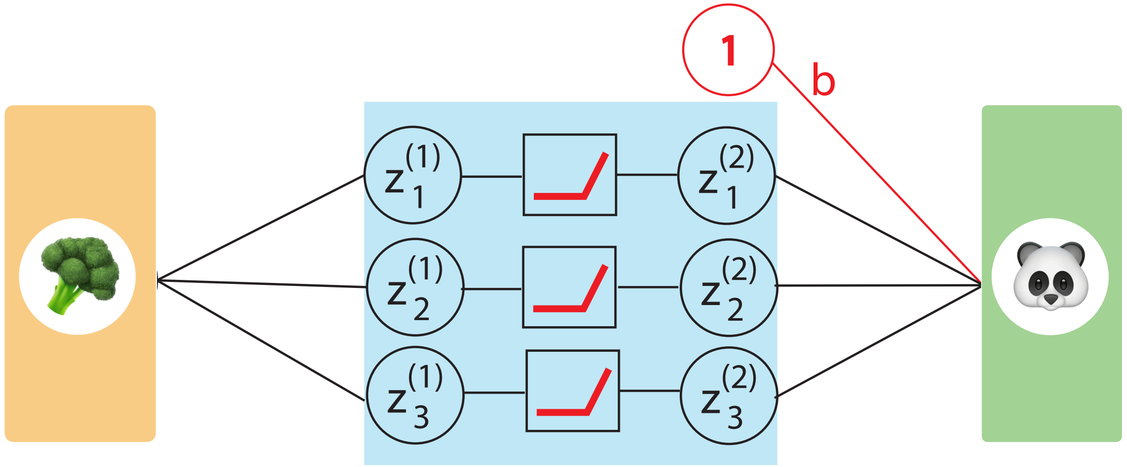


BNNs are powerful but…

- They are over-parameterized models (risk of overfitting, potential issues in extrapolation)
- Defining informative priors is not trivial (if at all possible)
- Estimated parameters are not directly interpretable
- They are not designed for hypothesis testing

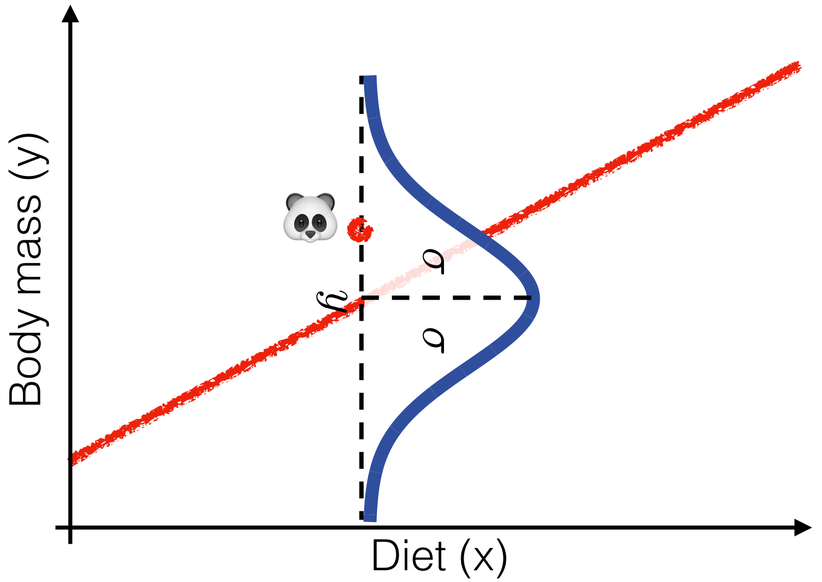
Training a supervised neural network

NN regression



Likelihood of an observed panda body mass

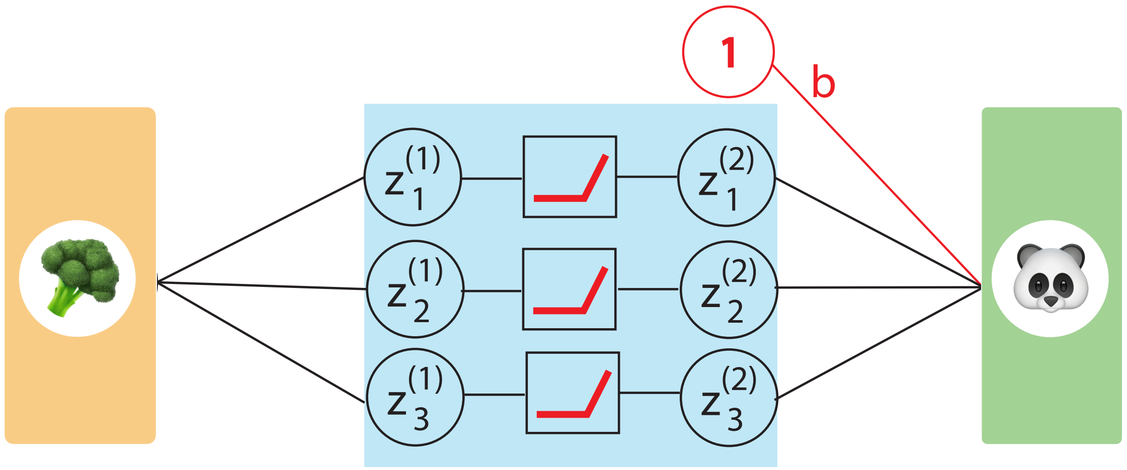
$$P(🐼|x, w, b, \sigma) \sim \mathcal{N}(y, \sigma)$$



Probability density function of a normal distribution

$$P(🐼|🟢, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(🐼 - 🟢)^2}{2\sigma^2}}$$

NN regression



Data ('training set')

Likelihood function: the normal density function

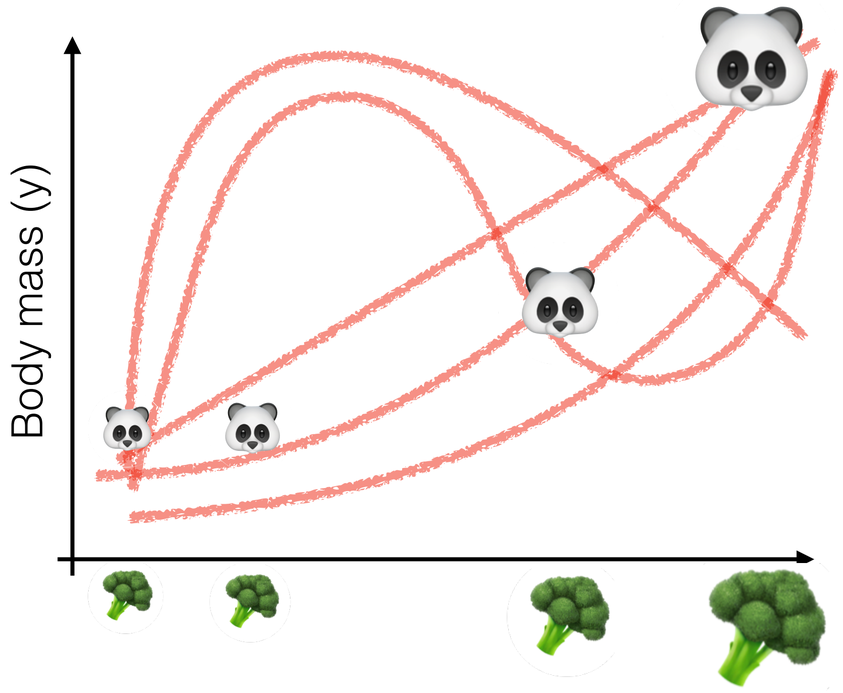
$$P(🐼 | y, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(🐼 - y)^2}{2\sigma^2}}$$

Loss function: mean squared error

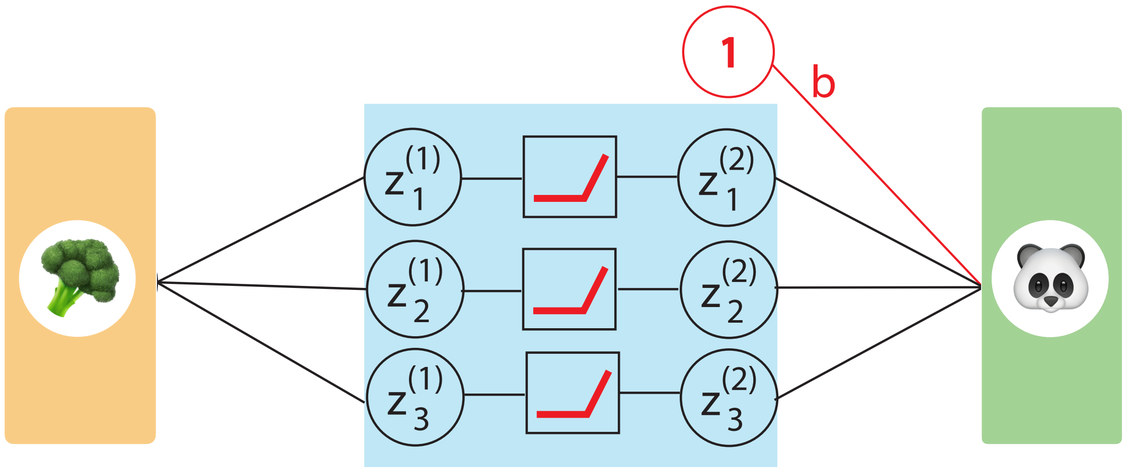
$$-\log P(🐼 | y) \propto (🐼 - y)^2$$

Squared difference between truth and prediction

Optimizing ('training') a model

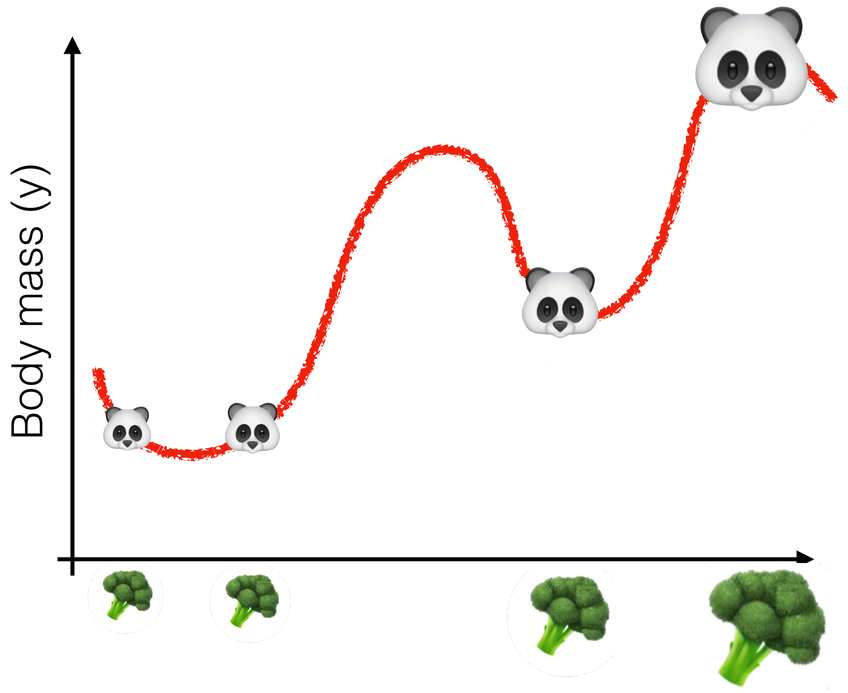


NN regression

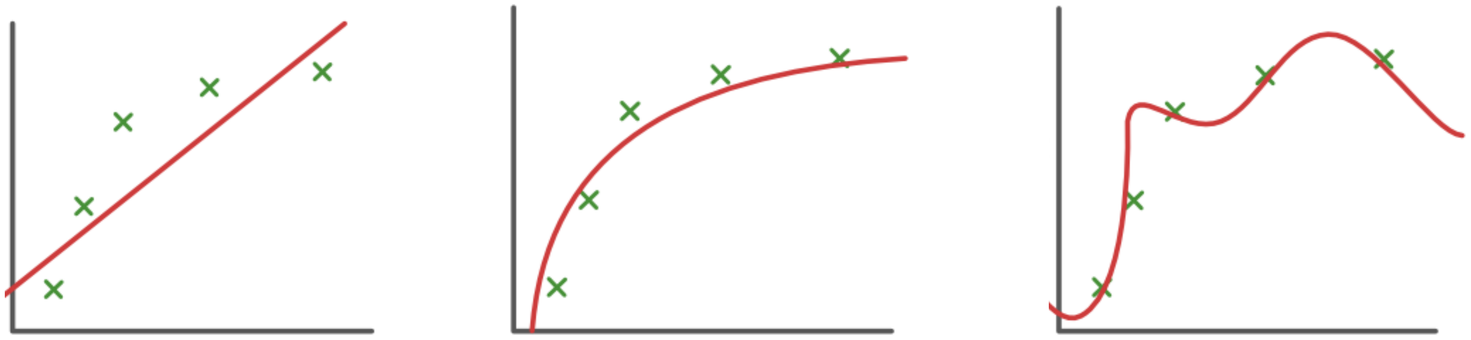


Data ('training set')

Best-fitting model



Neural networks are over-parameterized models
and will overfit with a maximum likelihood approach



Statistical inference:

Step 1: select best model while penalizing for complexity (e.g. AIC or Bayes factors)

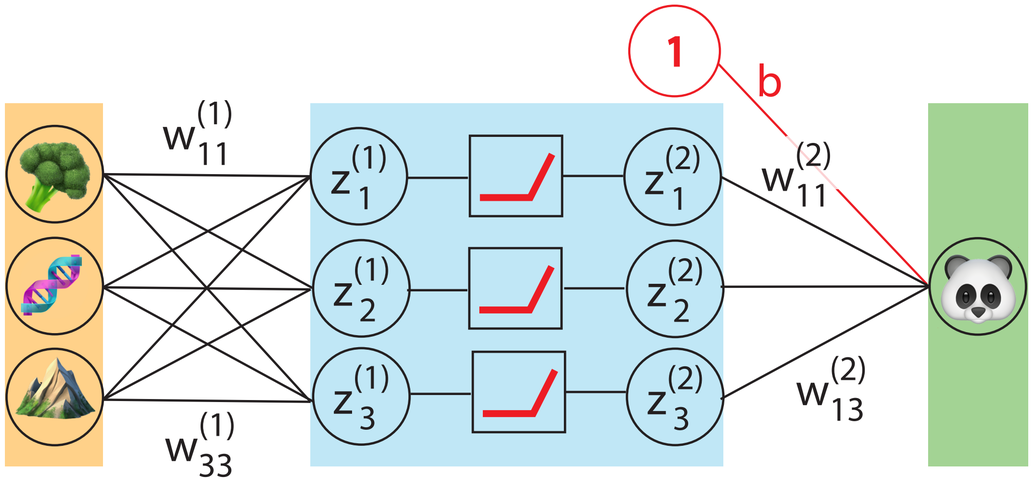
Step 2: find the parameters that maximize the likelihood of your data under the model

ML inference:

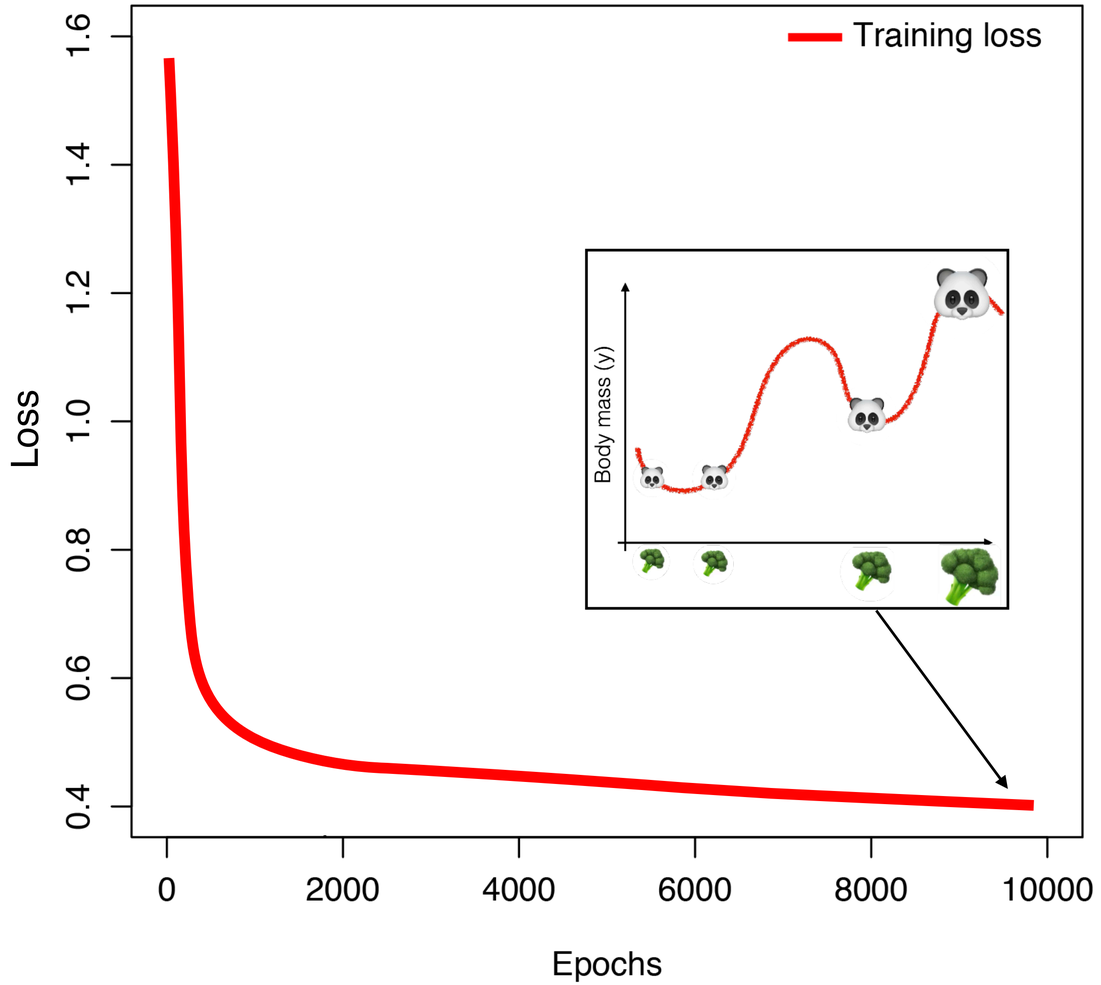
Step 1: fit your model while preventing overfitting (i.e. stop optimizing before reaching the maximum likelihood)

Step 2 (optional): find the hyper-parameters that give you the best results (e.g. based on a test set)

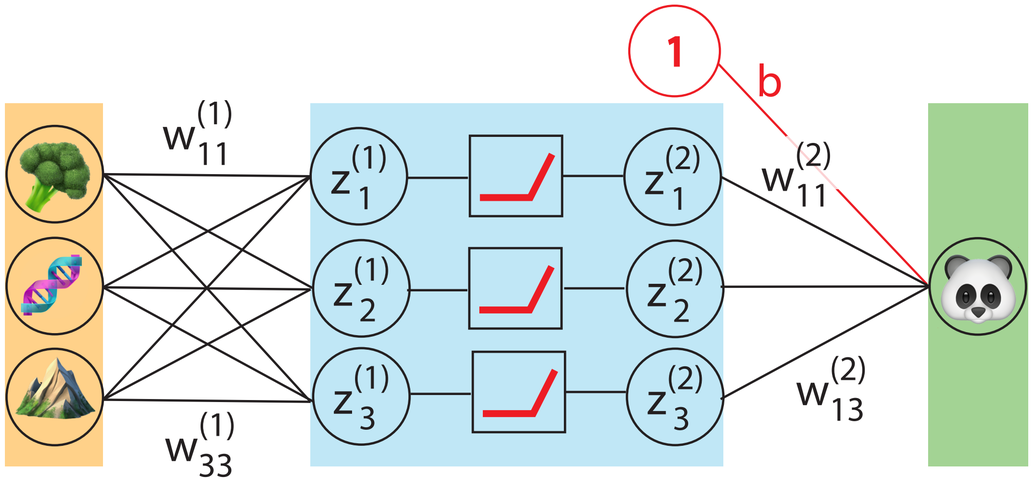
Optimization of a NN model



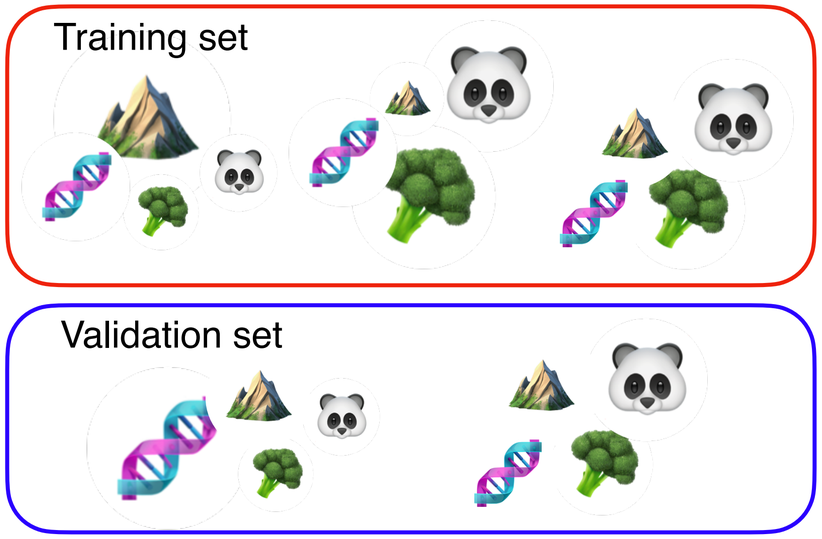
A maximum likelihood (or minimum loss) optimization of an NN would inevitably lead to over-fitting because NNs are over-parameterized

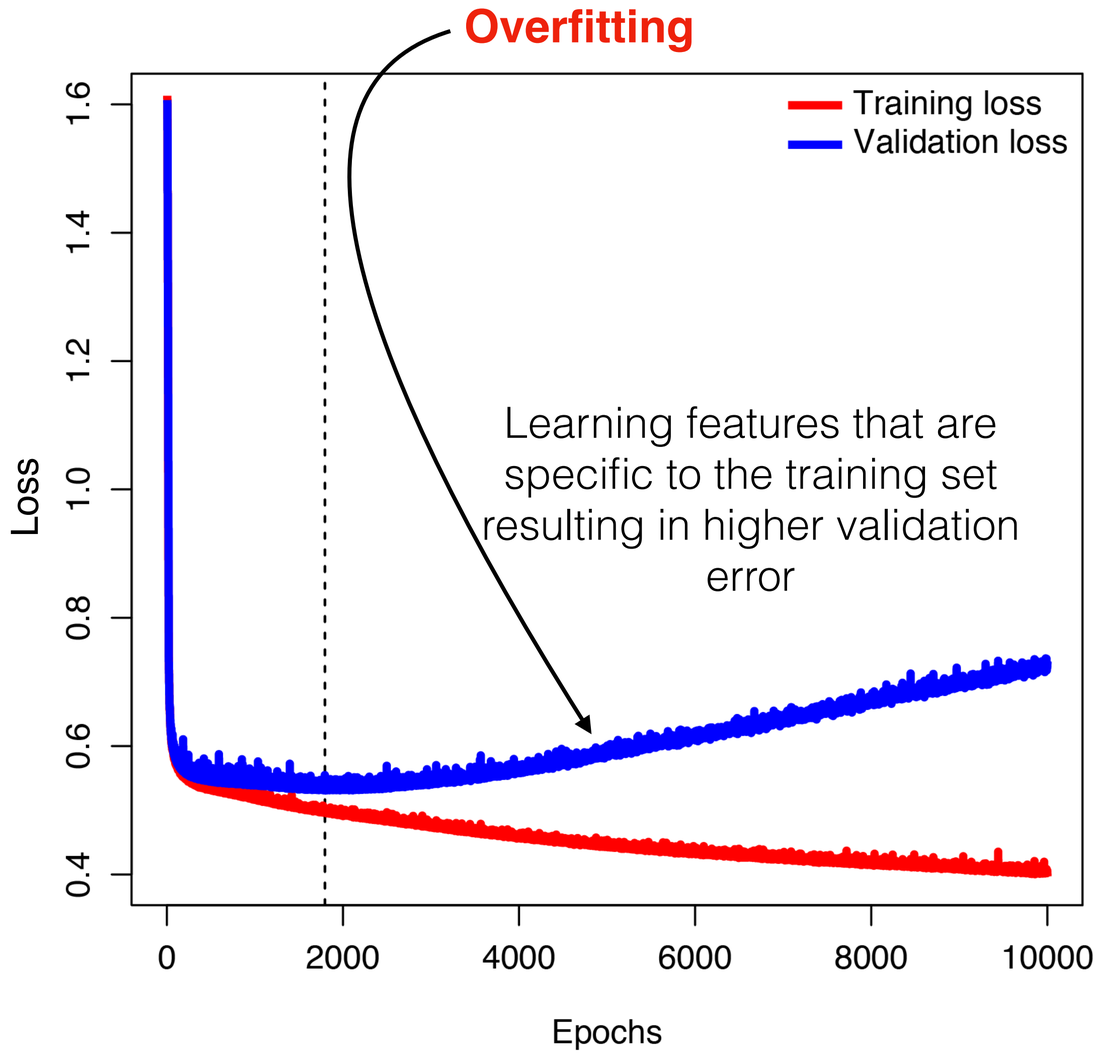


Optimization of a NN model

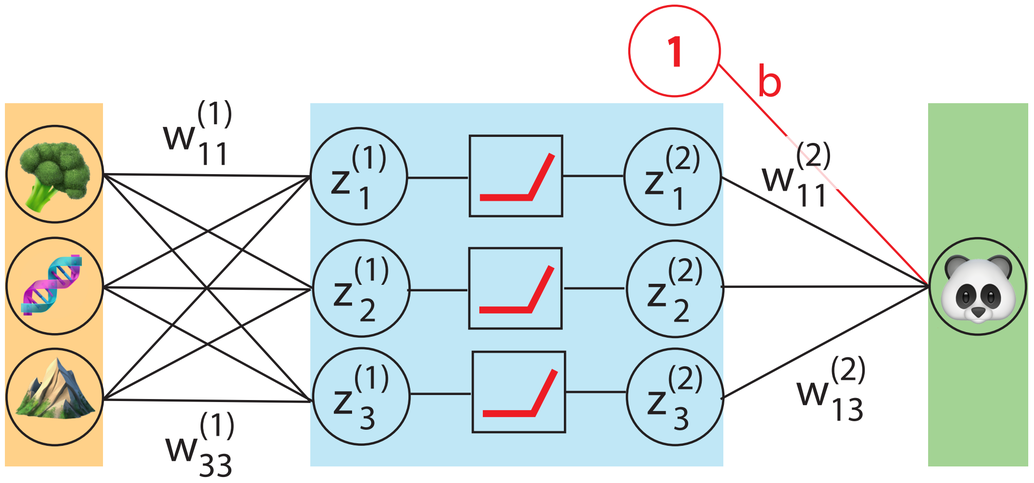


In machine learning a separate validation set is used to stop the optimization process before it starts overfitting

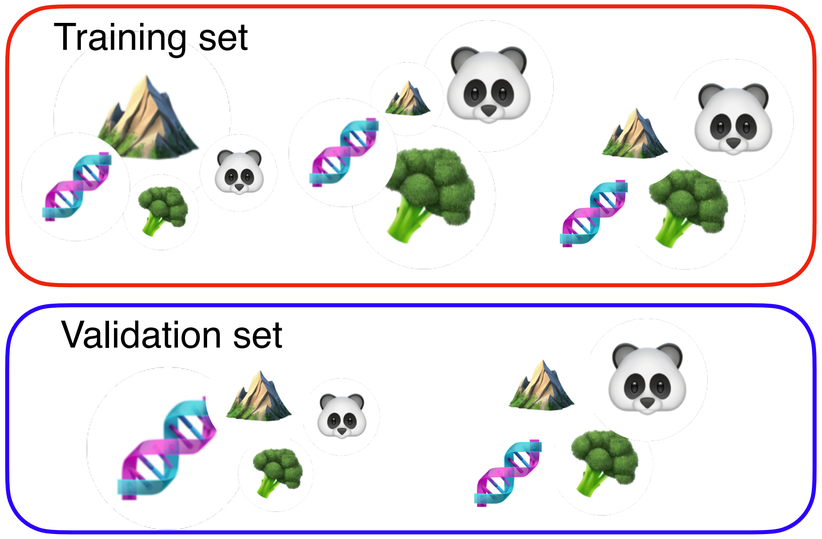


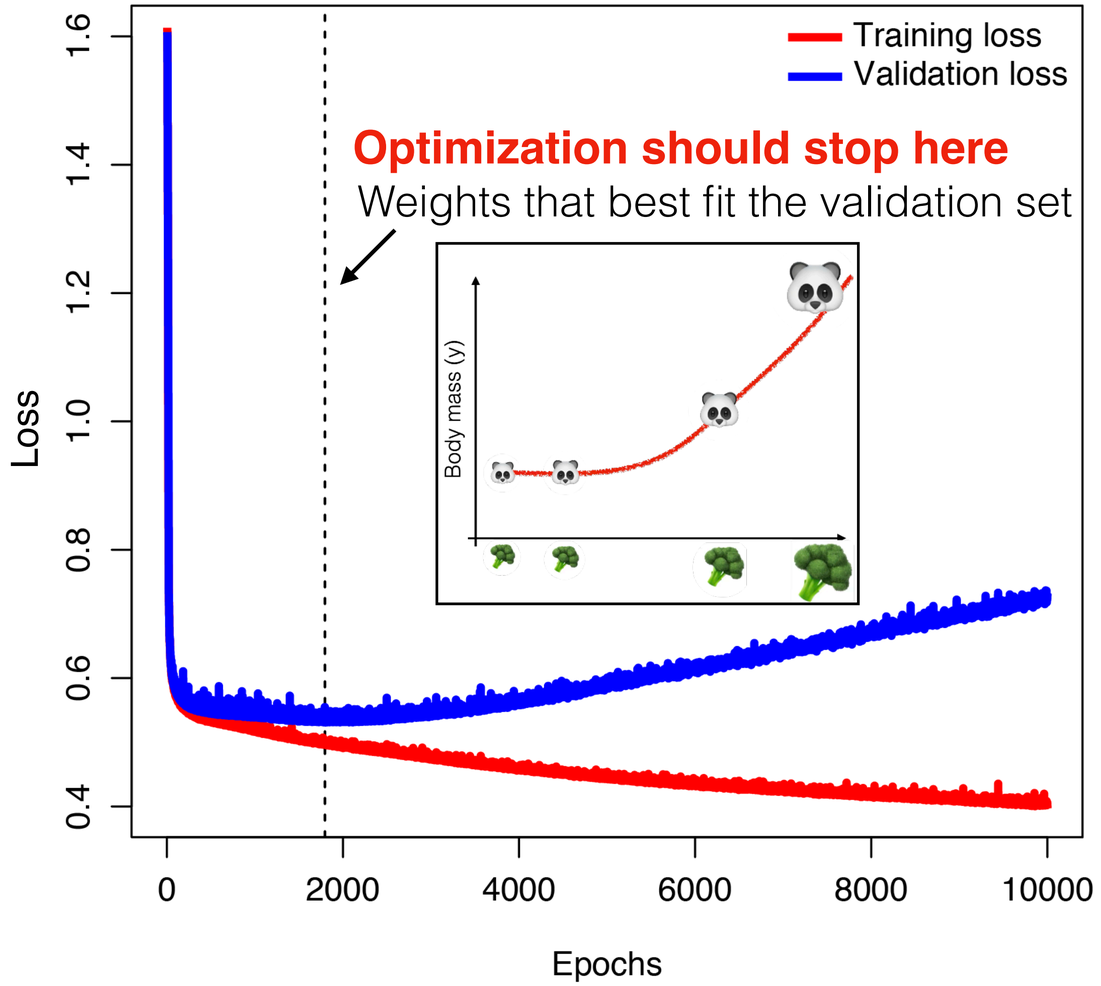


Optimization of a NN model

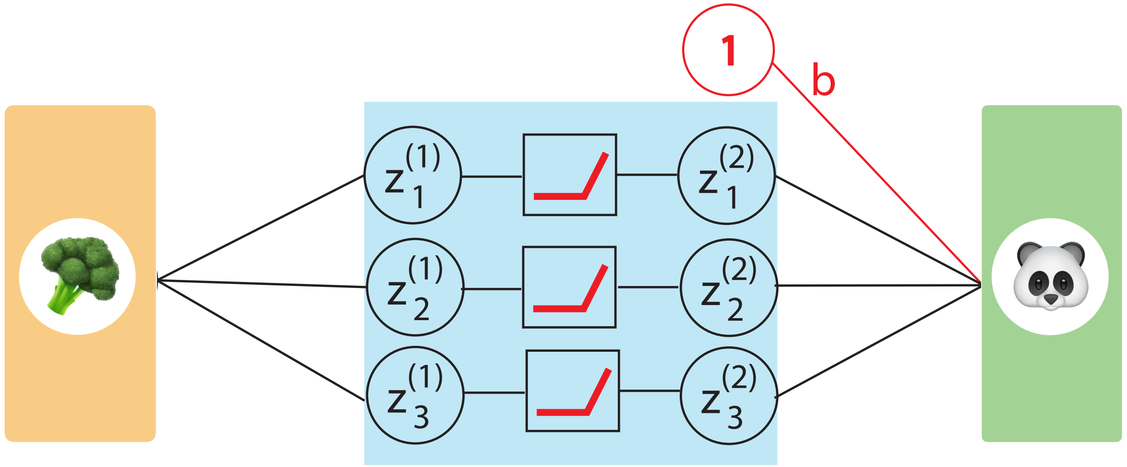


In machine learning a separate validation set is used to stop the optimization process before it starts overfitting





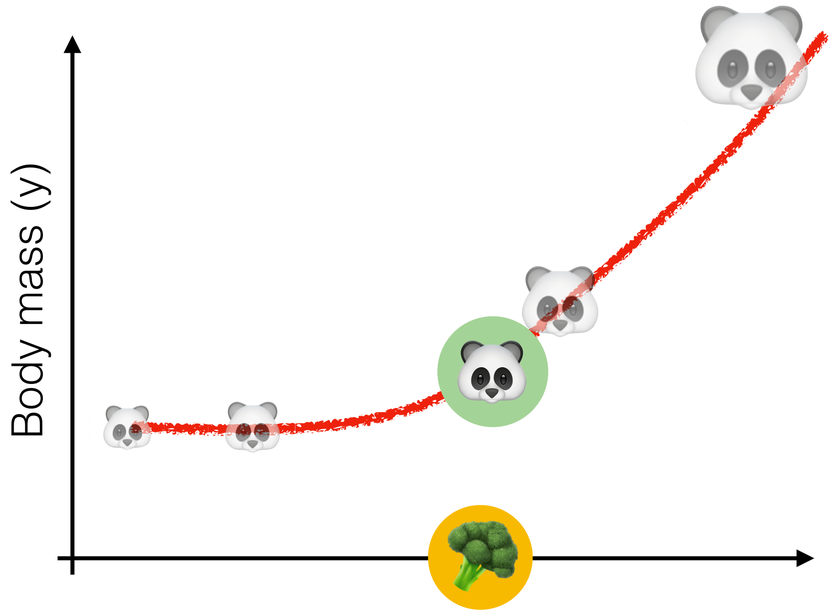
NN regression



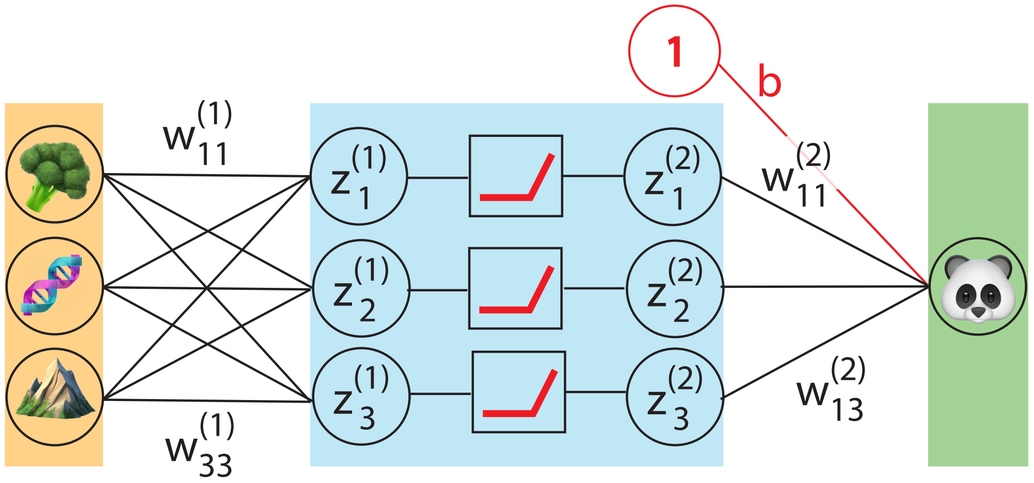
Data ('training set')

Predicting body mass from diet

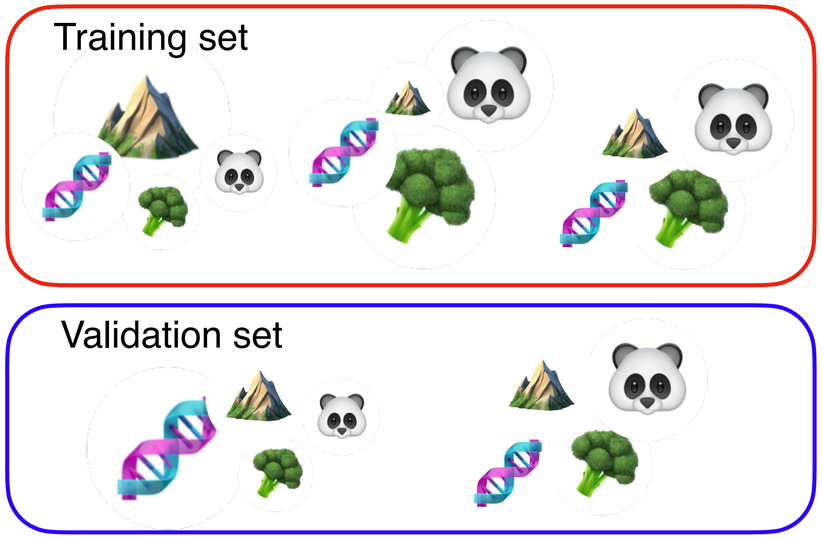
Trained model

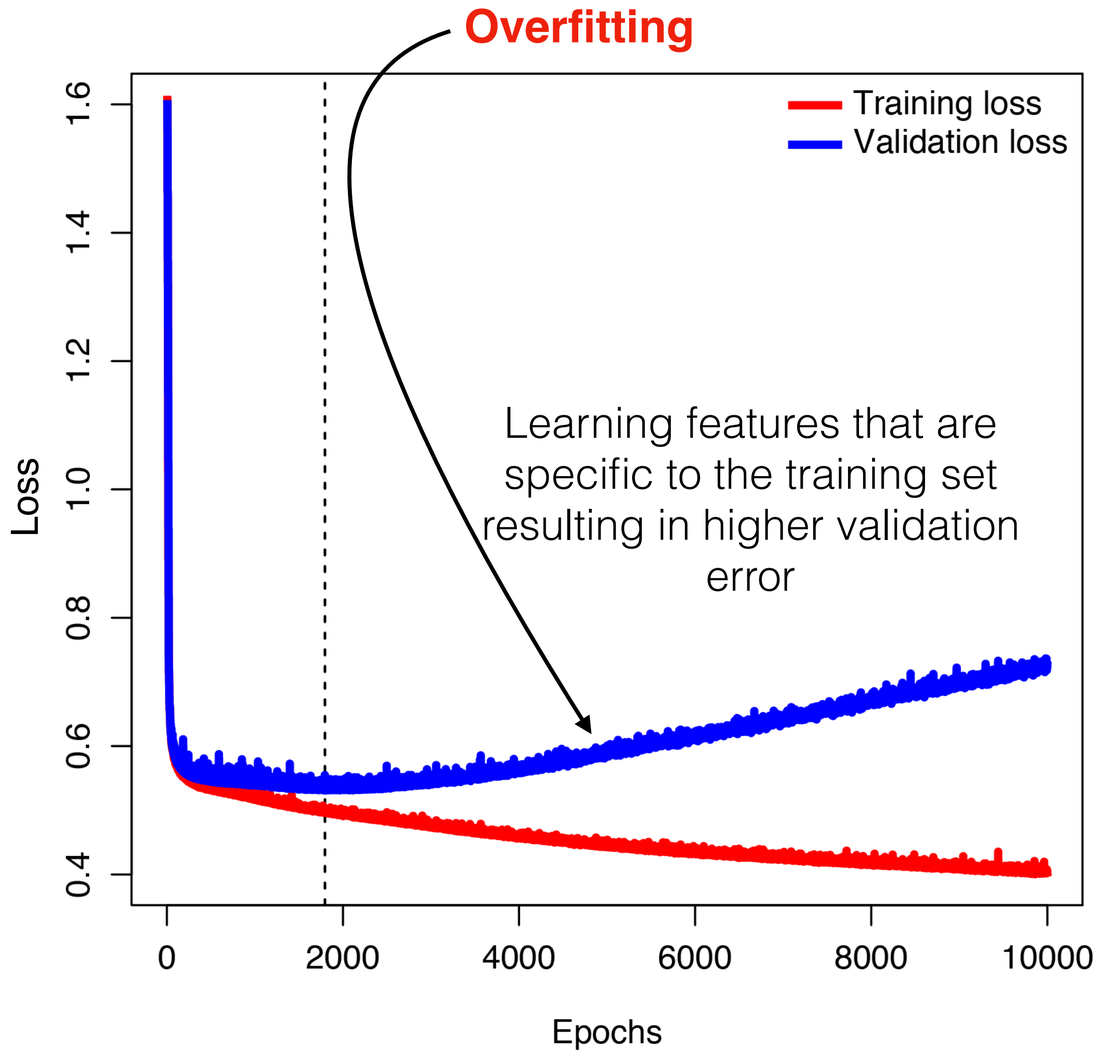


Optimization of a NN model

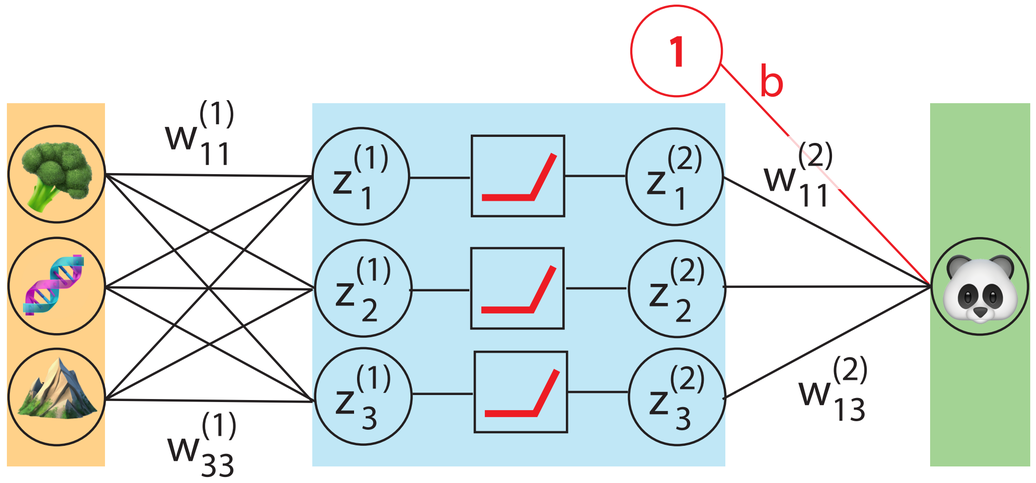


In machine learning a separate validation set is used to stop the optimization process before it starts overfitting

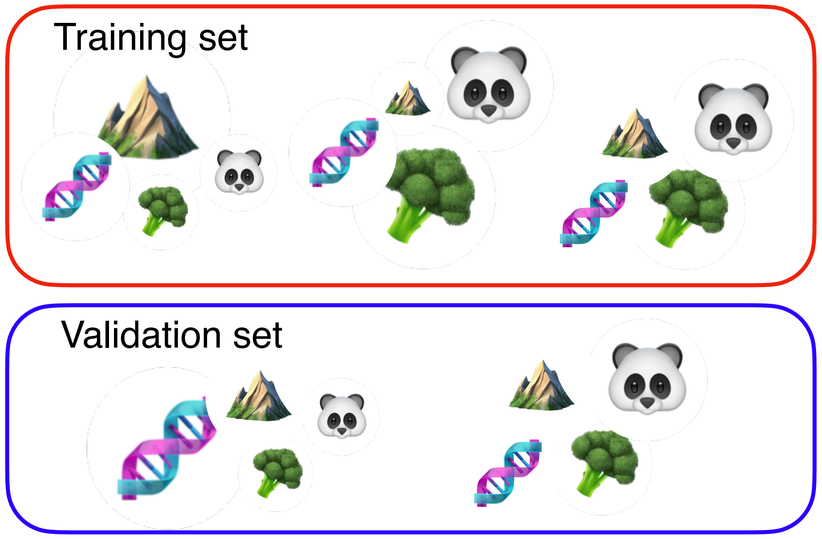


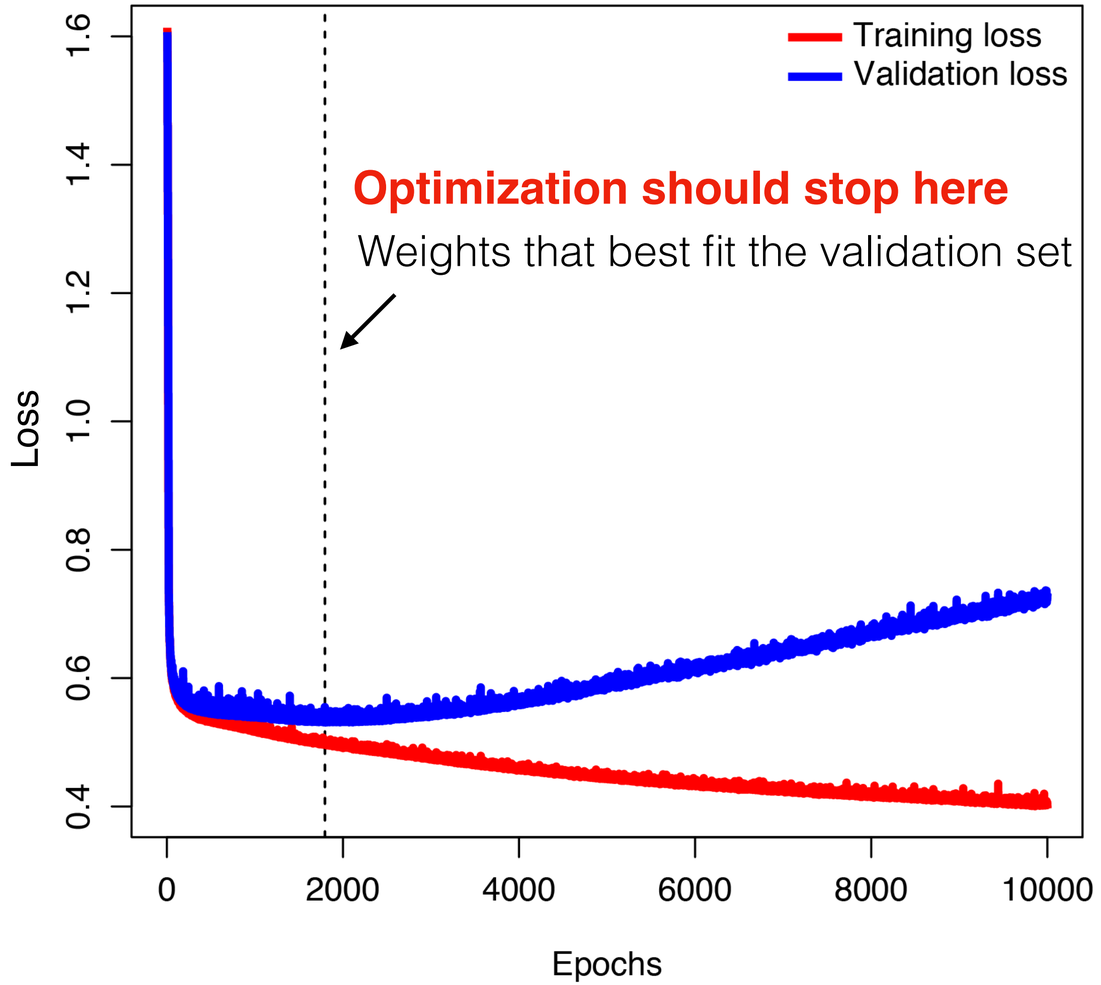


Optimization of a NN model

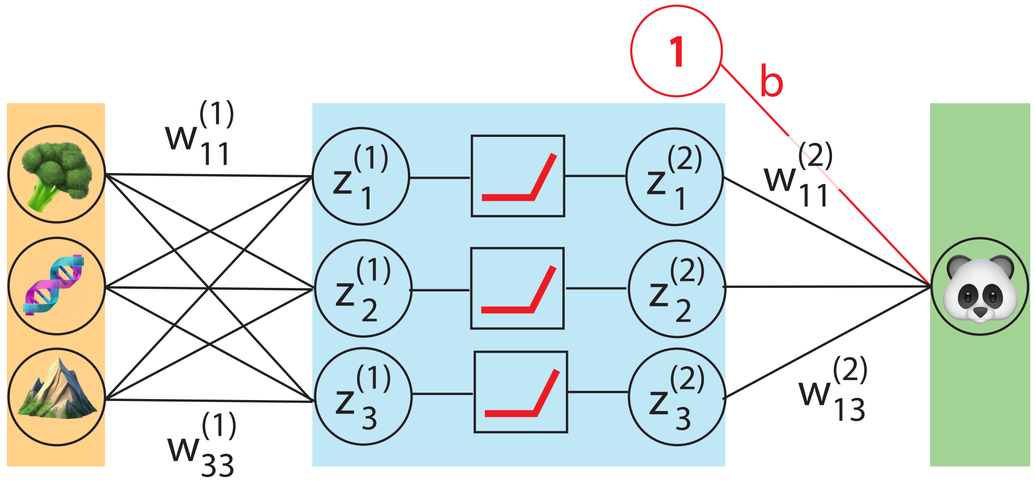


In machine learning a separate validation set is used to stop the optimization process before it starts overfitting





Likelihood in a neural network model of classification

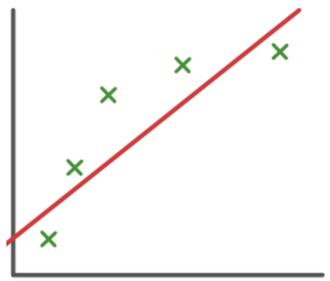


In a BNN the priors have a regularizing effect that limits the risk of overfitting

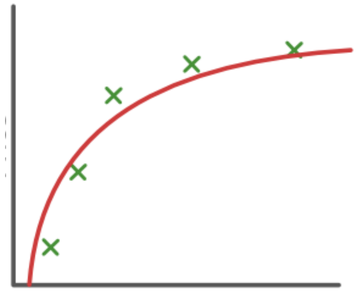
Priors on the weights

$$P(w) \sim \mathcal{N}(0, 1)$$

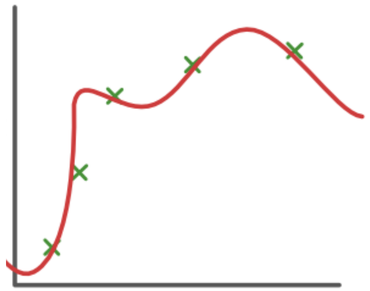
Under-fitting

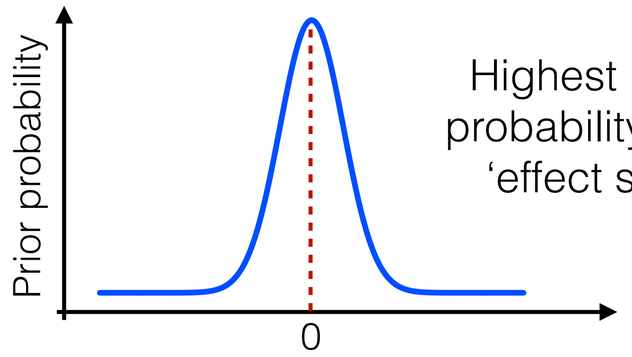


Just right



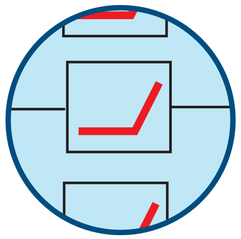
Over-fitting



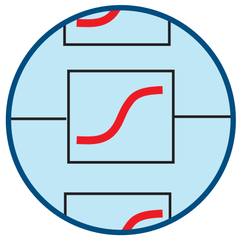


Parameterization of a neural network

Activation functions



ReLU: rectified linear unit

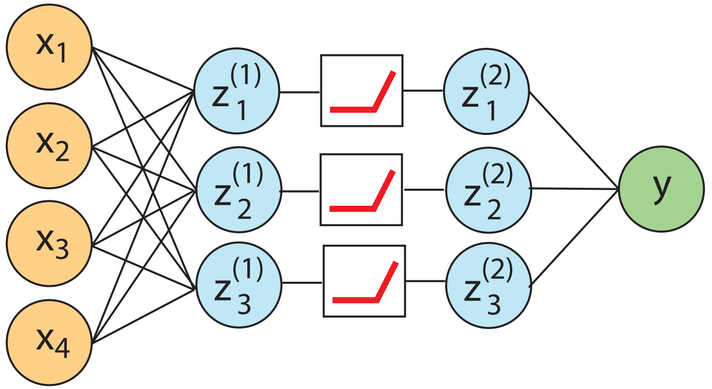


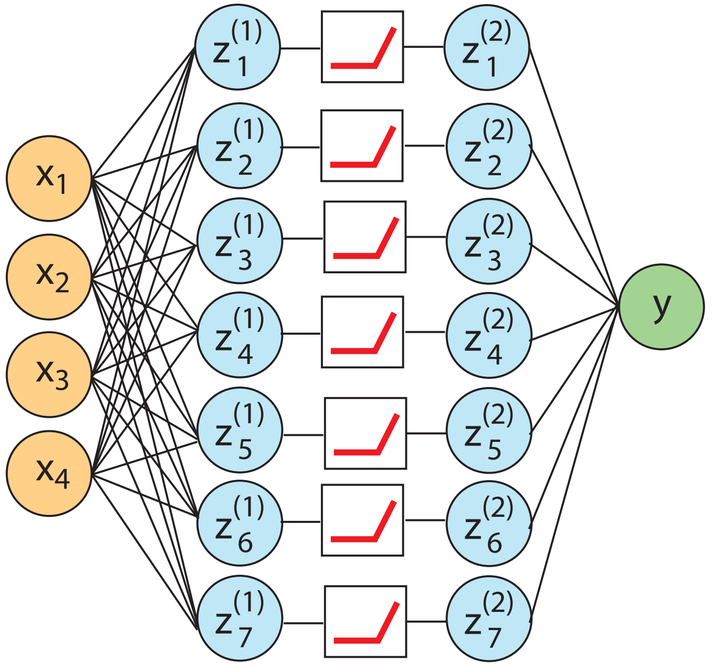
Sigmoid

```
ReLU(x) = max(0, x)
```

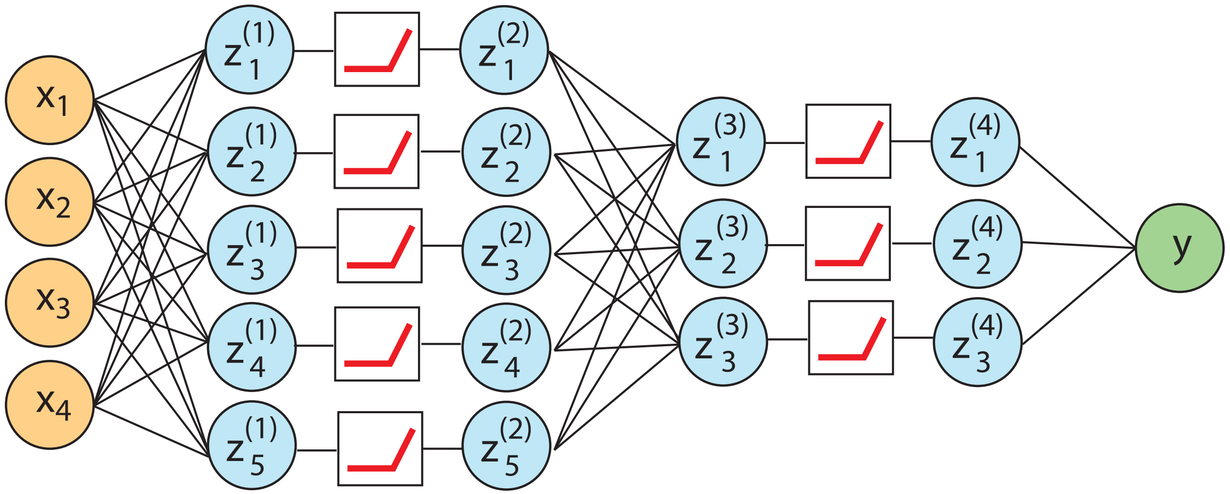
```
sigmoid(x) = 1 / (1 + exp(-x))
```

Number of nodes





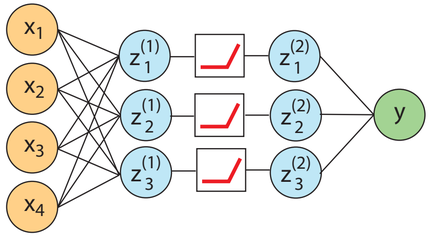
Number of hidden layers (deep NNs)

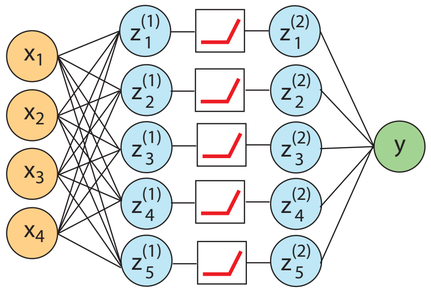


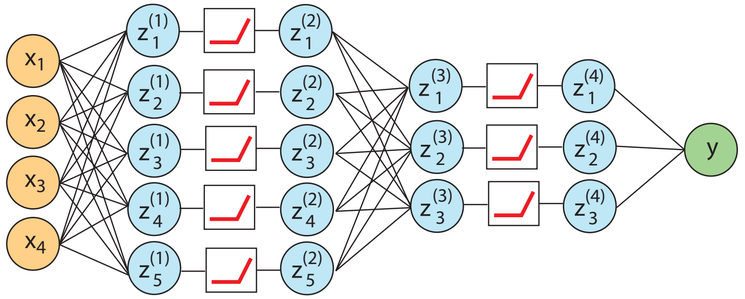
Parameterization of a neural network: how to choose?

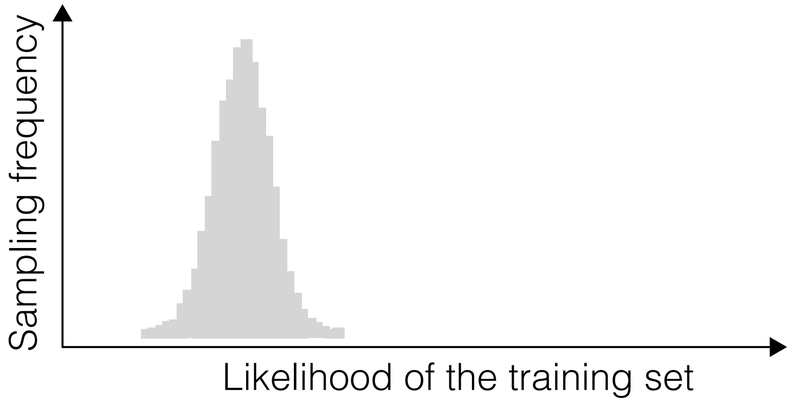
Training set

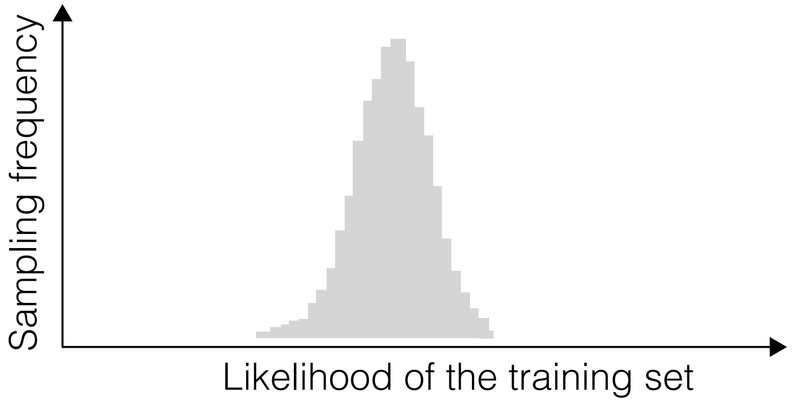
Validation set

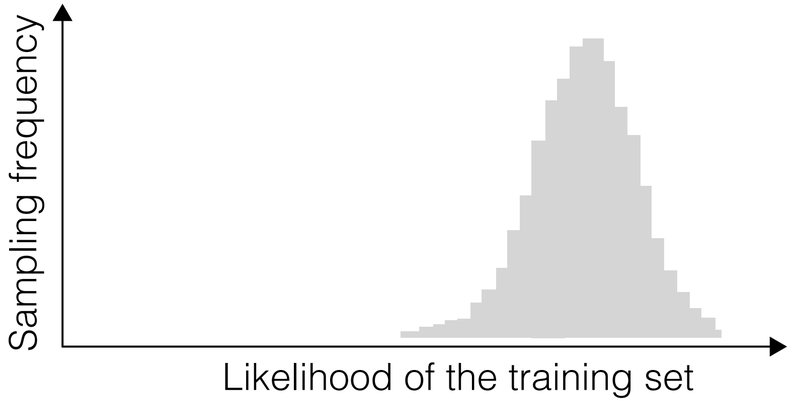








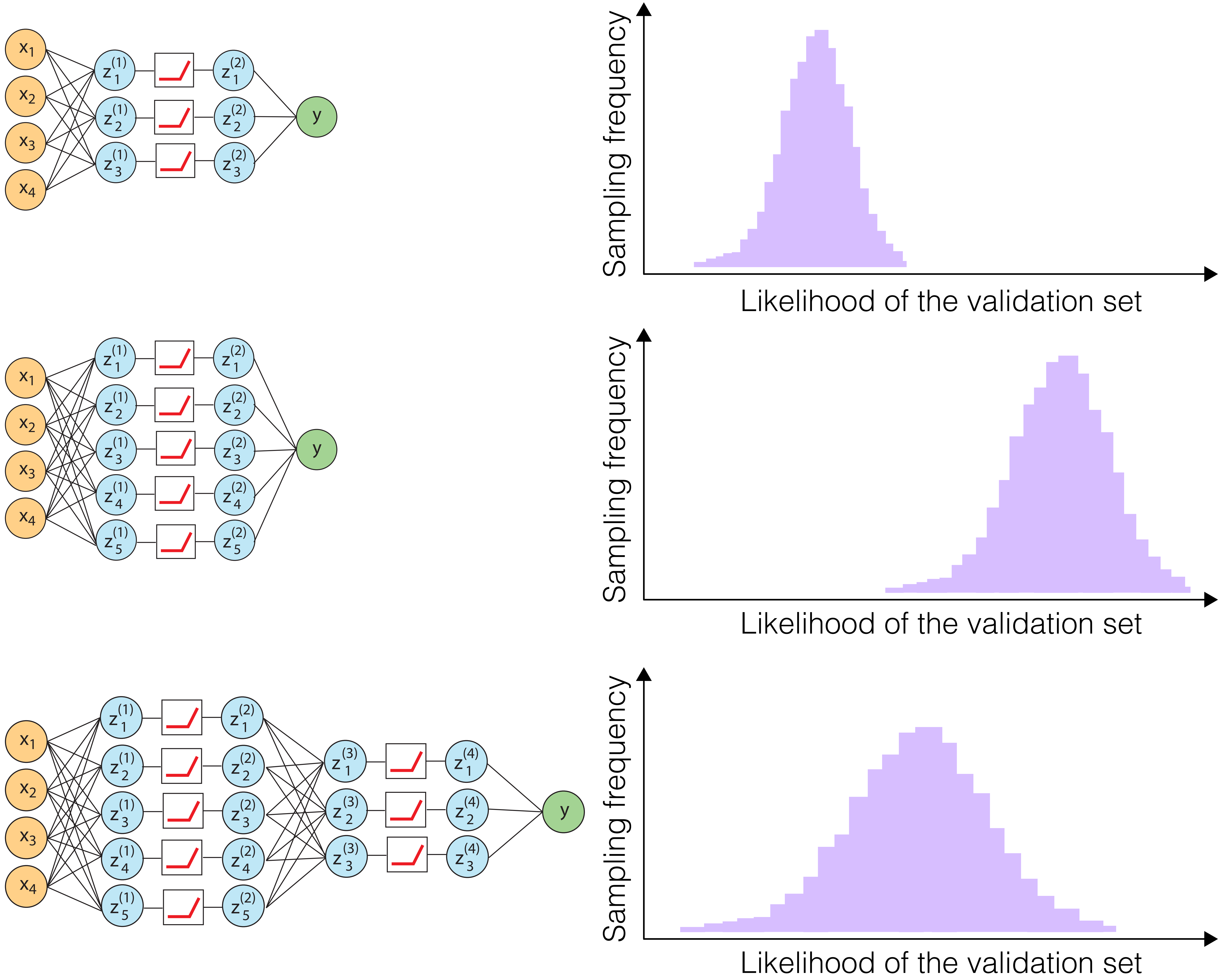




Parameterization of a neural network: how to choose?

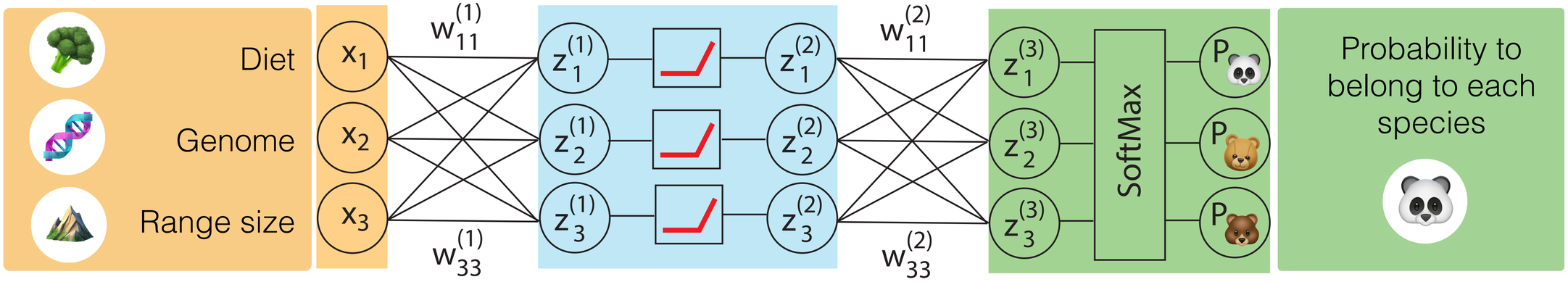
Training set

Validation set



Neural networks for classification

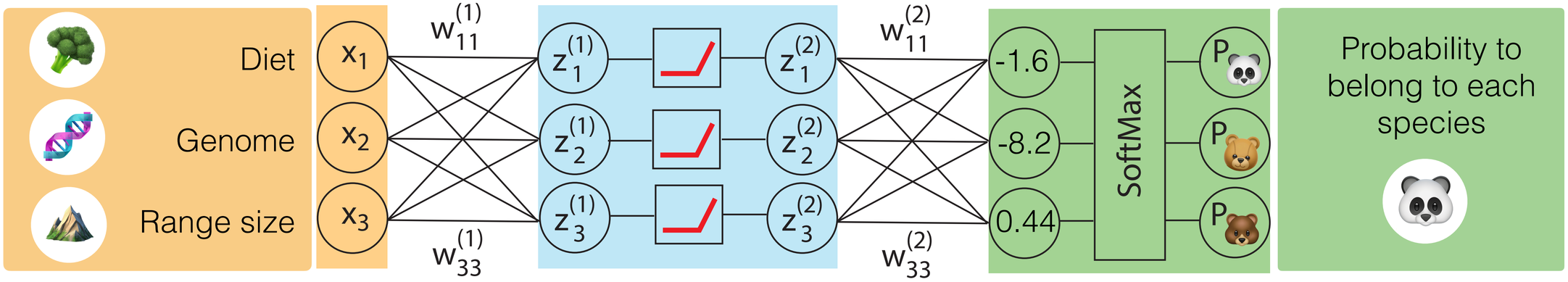
Neural networks for classification



Classification task

Find the parameter for any combination of diet, genome and range size assign the highest probability to the correct species

Neural networks for classification



SoftMax function

Convert arbitrary real values into probabilities

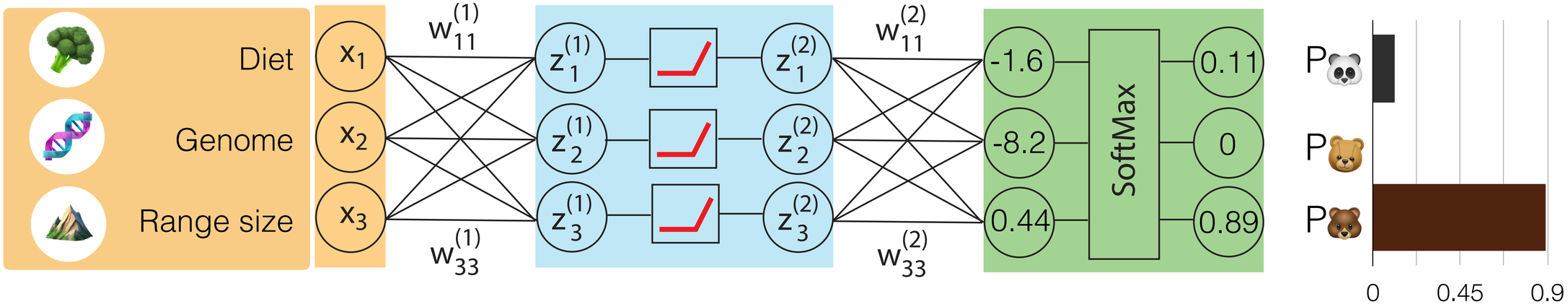
$$\begin{pmatrix} -1.64 \\ -8.18 \\ 0.44 \end{pmatrix}$$

In Python:

```
y = np.exp(z3) / np.sum(np.exp(z3))
```

$z^{(3)}$

Neural networks for classification



SoftMax function

Convert arbitrary real values into probabilities

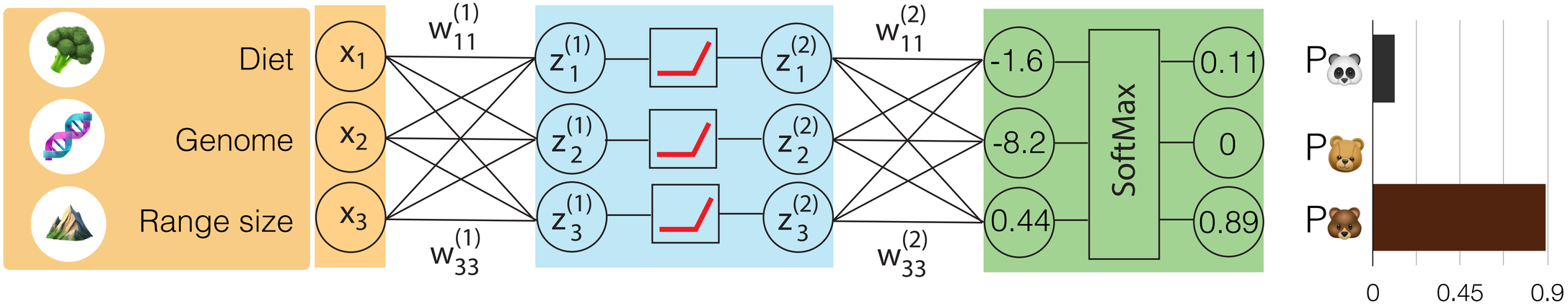
In Python:

```
y = np.exp(z3) / np.sum(np.exp(z3))
```

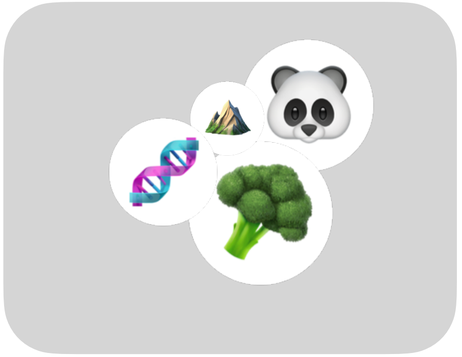
$$\begin{pmatrix} -1.64 \\ -8.18 \\ 0.44 \end{pmatrix} \rightarrow \begin{matrix} \frac{e^{-1.64}}{(e^{-1.64}+e^{-8.18}+e^{0.44})} \\ \frac{e^{-8.18}}{(e^{-1.64}+e^{-8.18}+e^{0.44})} \\ \frac{e^{0.44}}{(e^{-1.64}+e^{-8.18}+e^{0.44})} \end{matrix} = \begin{pmatrix} 0.11 \\ 0 \\ 0.89 \end{pmatrix}$$

$z^{(3)}$ — **SoftMax** — P

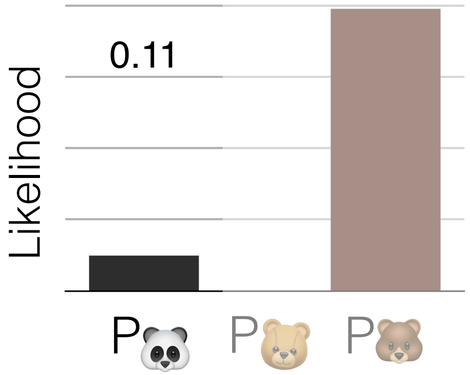
Neural networks for classification



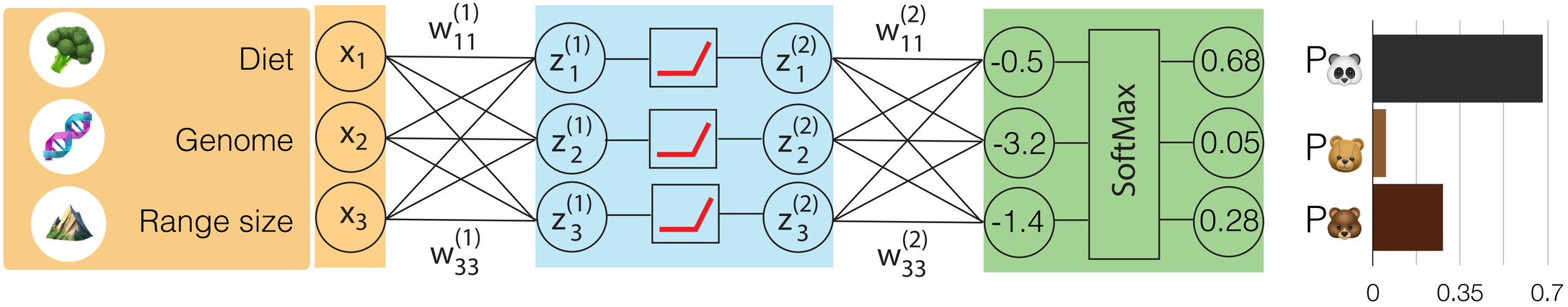
Likelihood function in a classification model



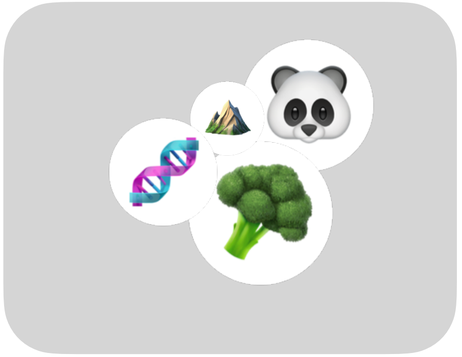
Probability mass function of a categorical distribution



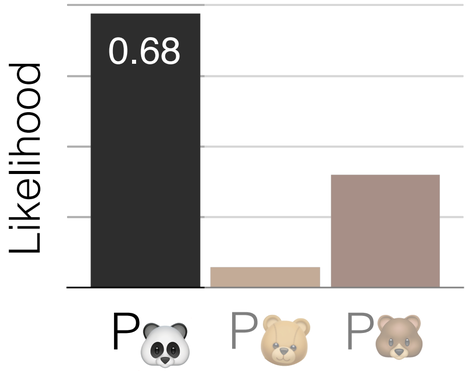
Neural networks for classification



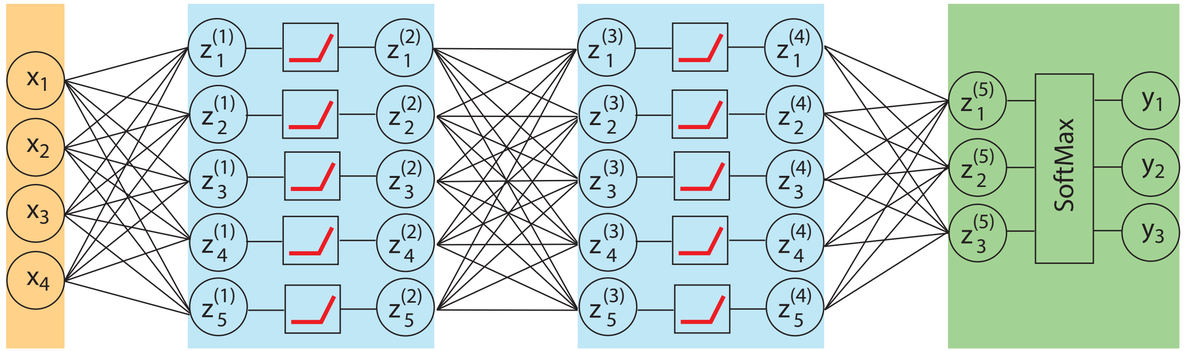
Likelihood function in a classification model



Probability mass function of a categorical distribution



Likelihood in a neural network model of classification

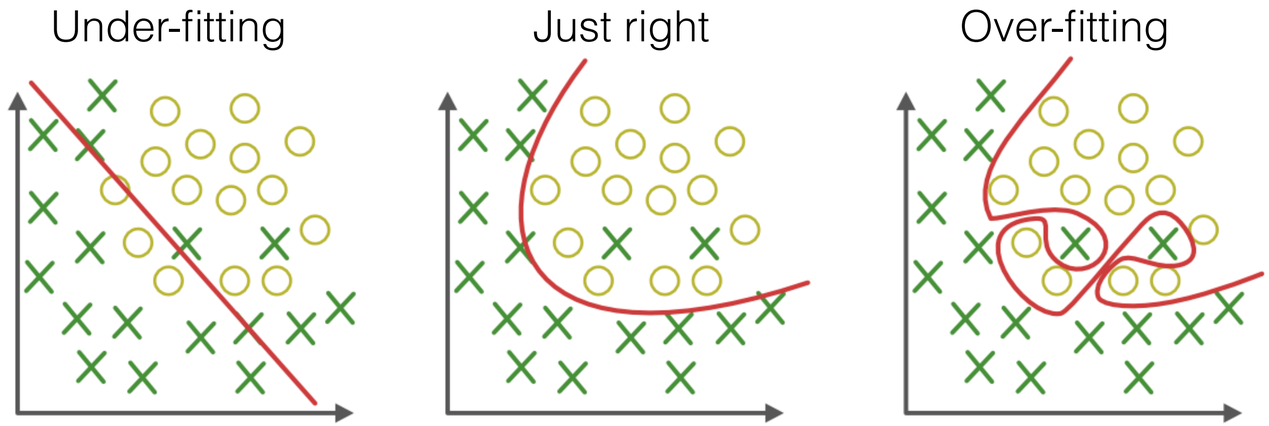


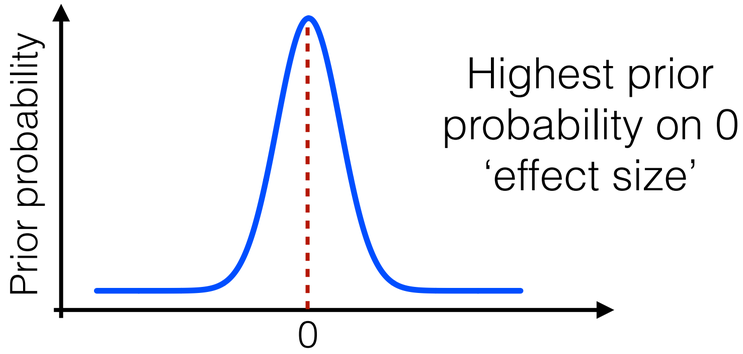
A maximum likelihood optimization of an NN would inevitably lead to over-fitting because NNs are over-parameterized

In a BNN the priors have a regularizing effect that limits the risk of overfitting

Priors on the weights

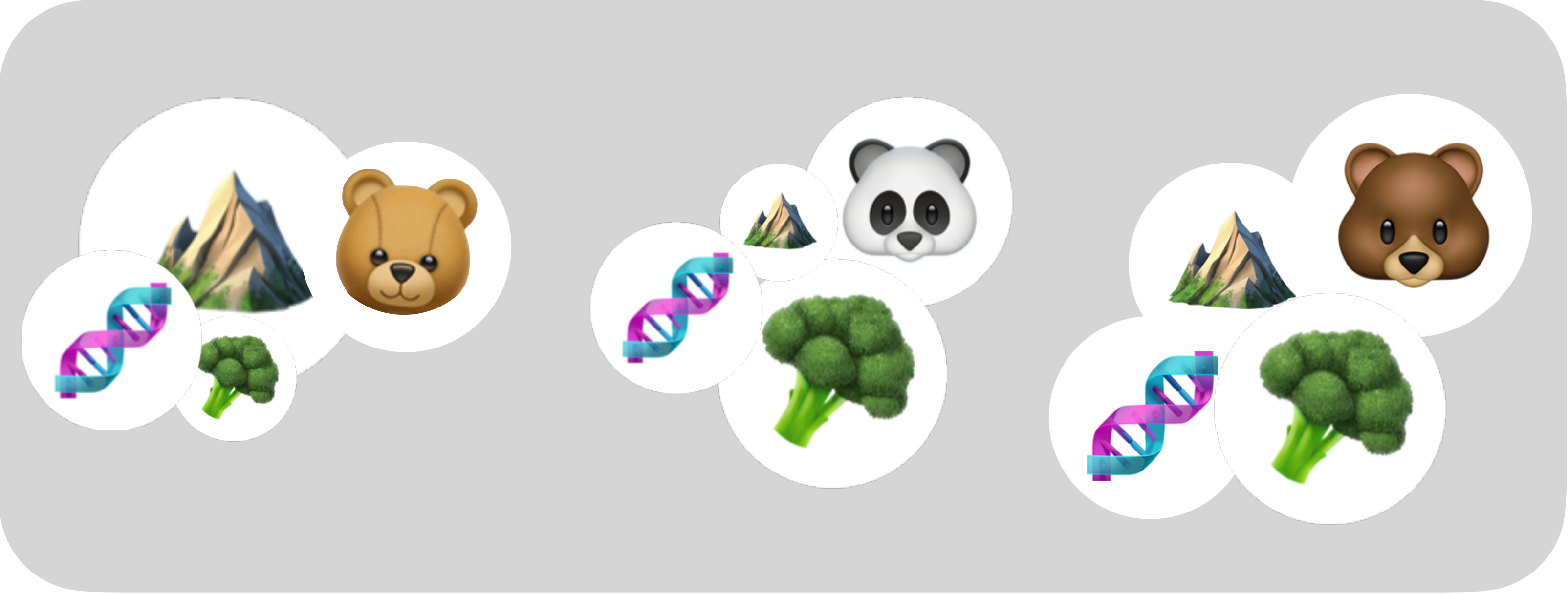
$$P(w) \sim \mathcal{N}(0, 1)$$



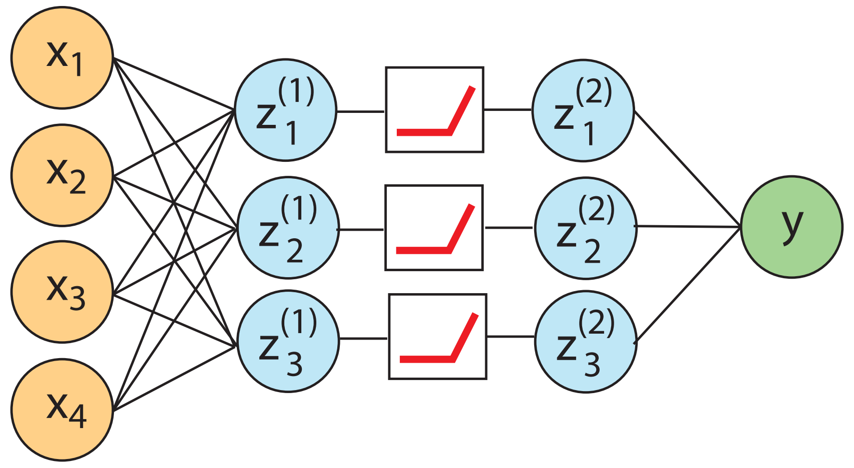


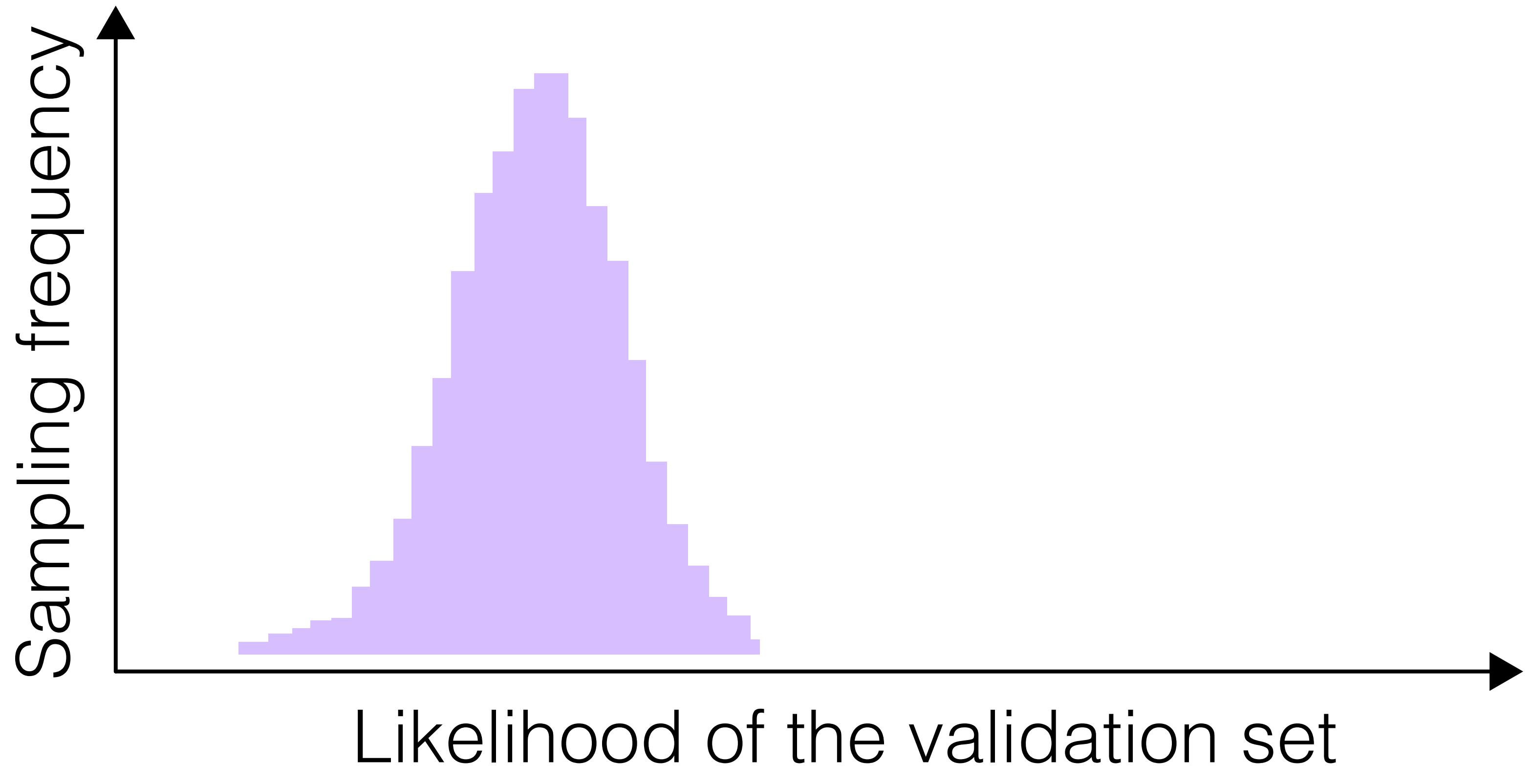
Parameterization of a neural network: how to choose?

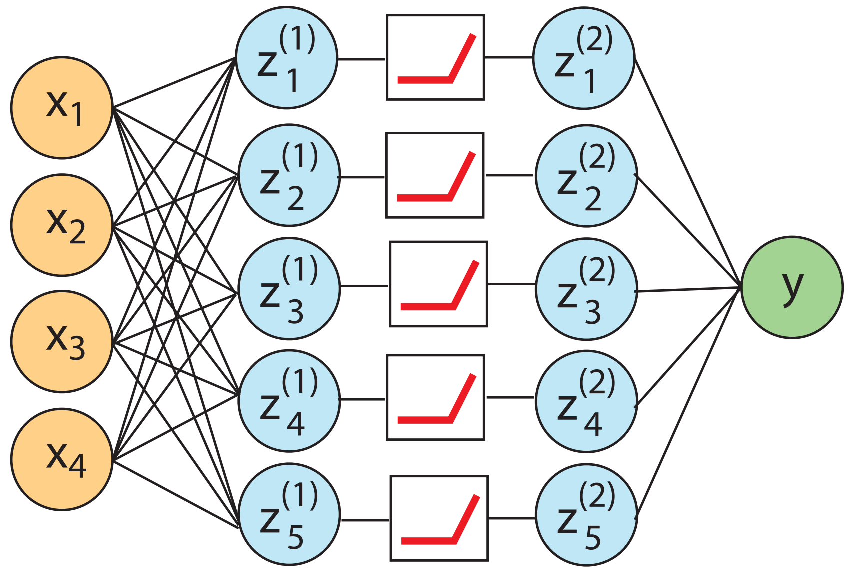
Training set

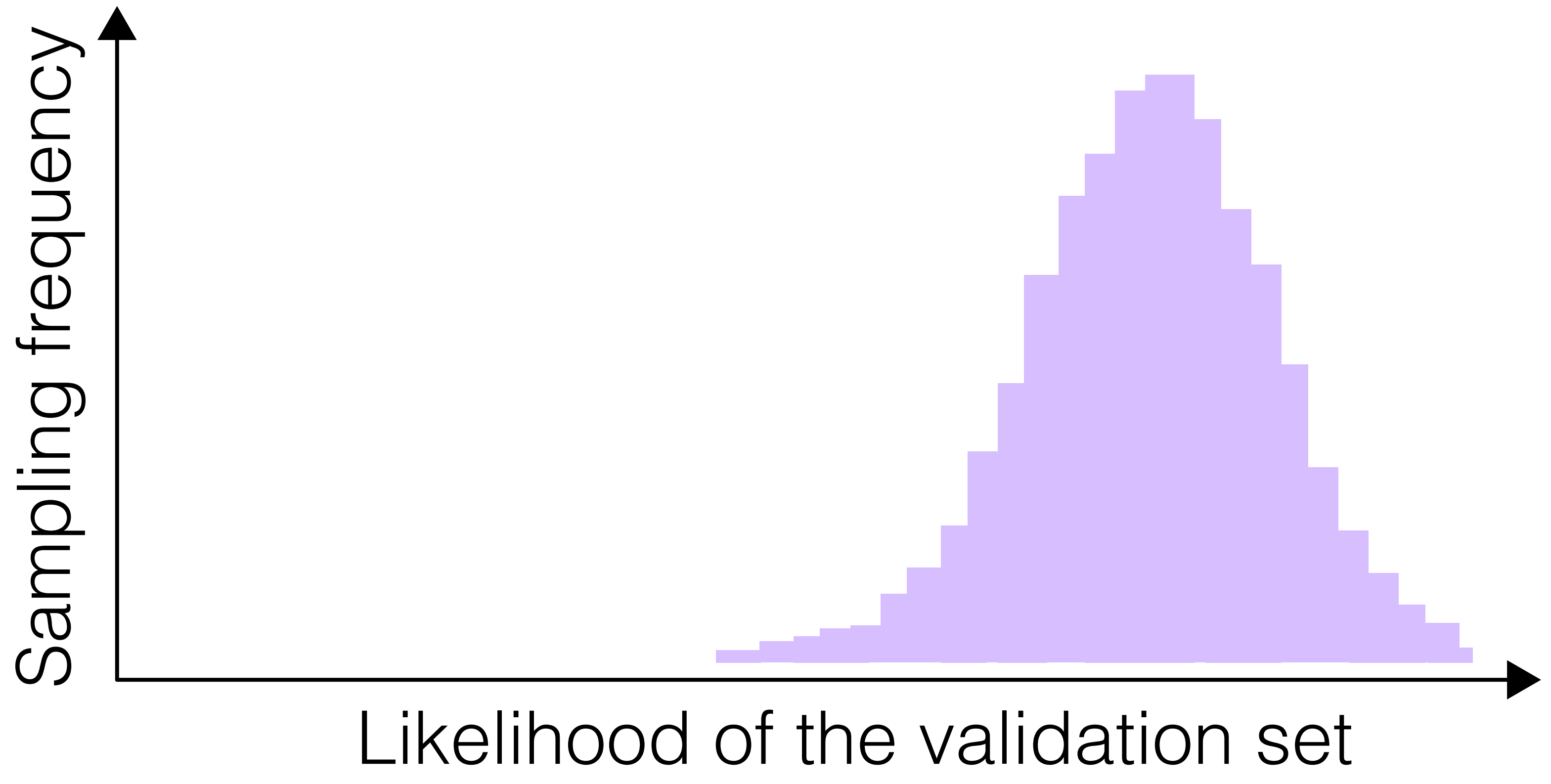


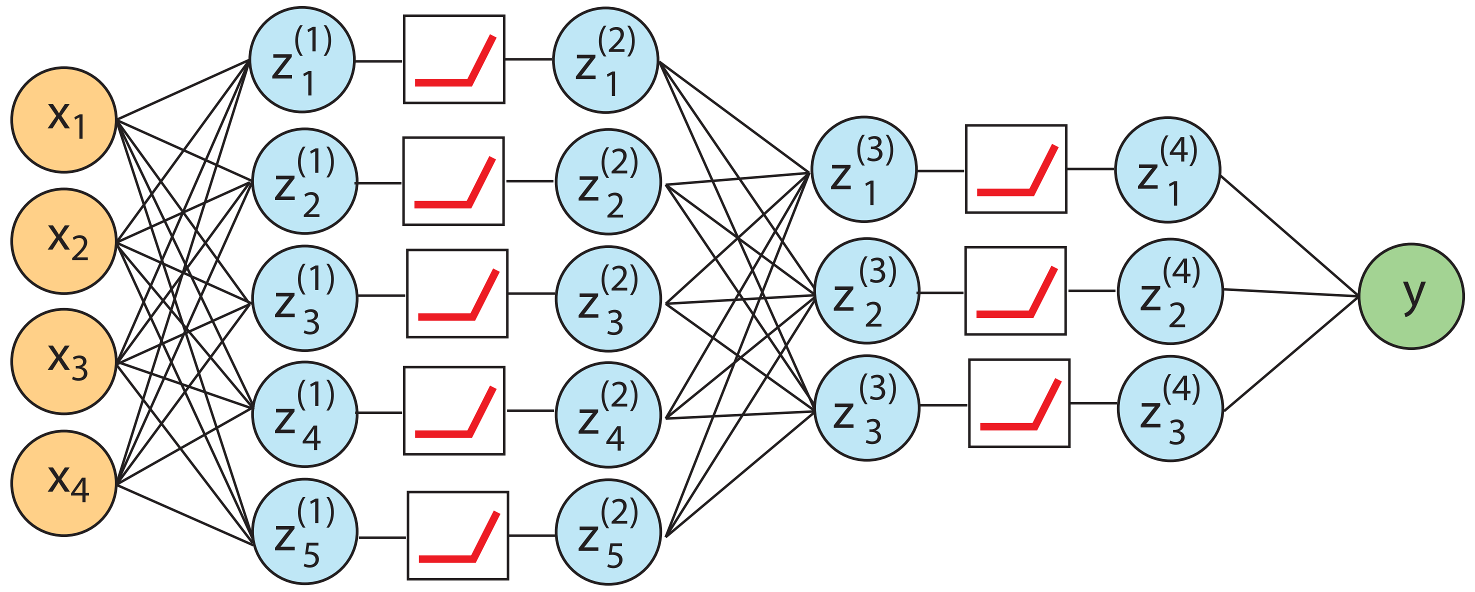
Validation set

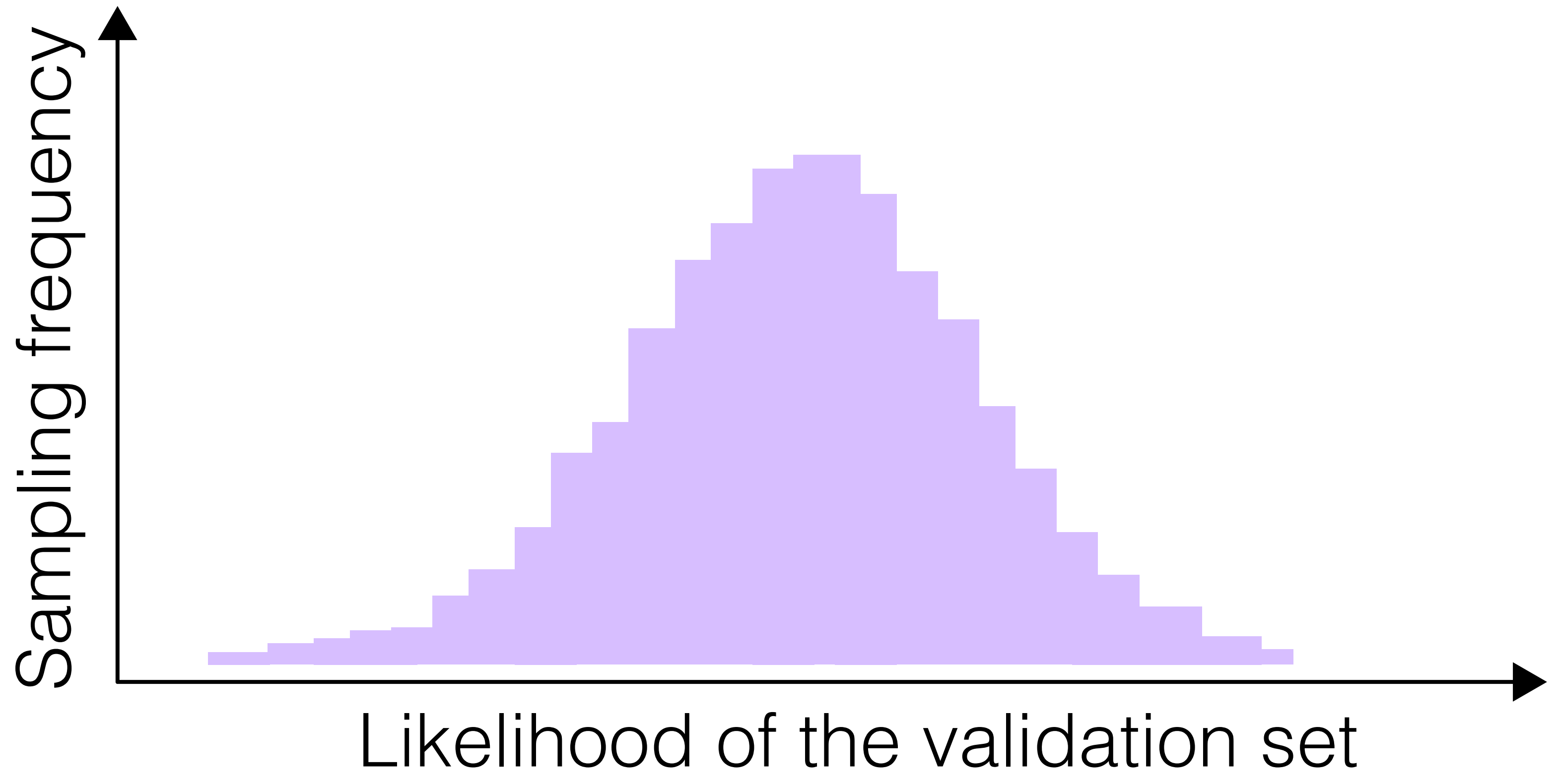






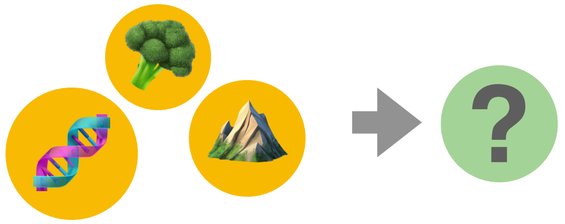




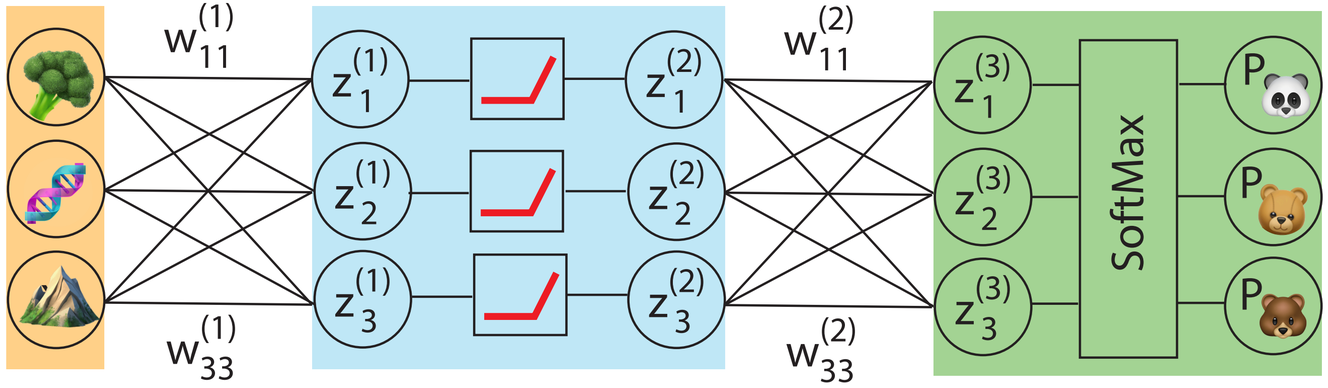


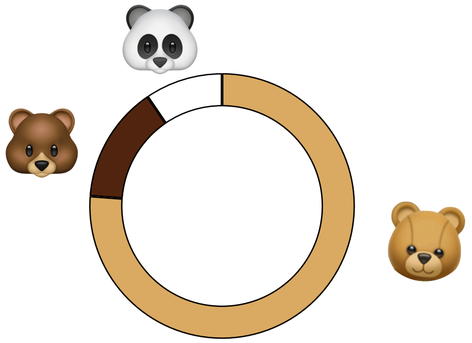
Predictions using a trained BNN

Predicting body mass from diet, genome, range size

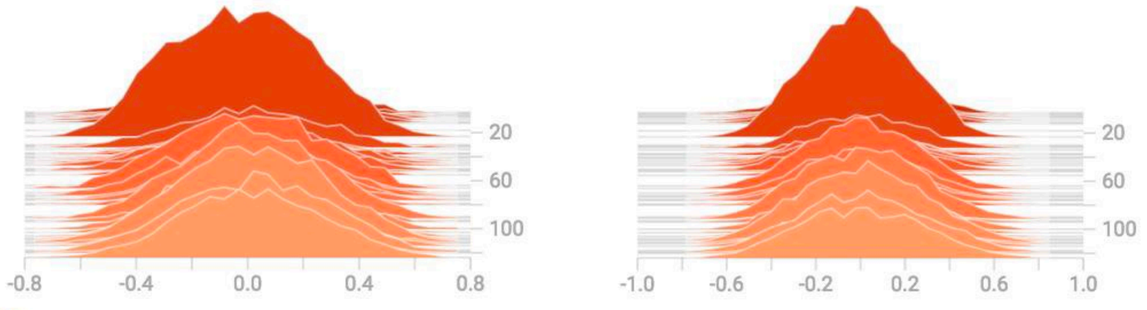


Ground truth





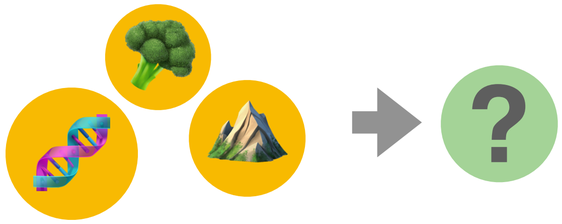
Posterior samples of the weights



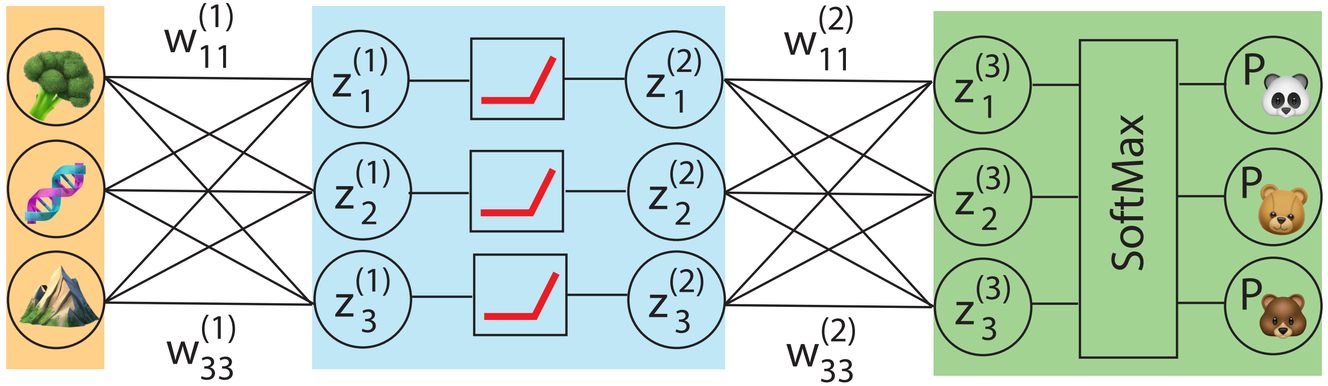
Posterior probability associated to each class

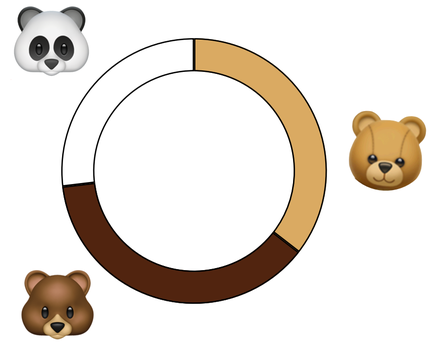
Predictions using a trained BNN – identifying unknowns

Predicting body mass from diet, genome, range size

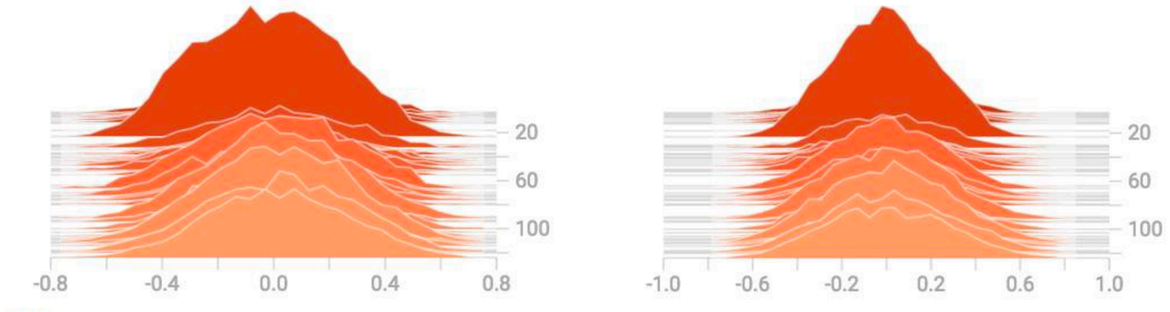


Ground truth: out-of-distribution!





Posterior samples of the weights



Low posterior probabilities among classes

Model performance: classification accuracy

Accuracy: $\frac{33}{37} = 89\%$

Error rate: $\frac{4}{37} = 11\%$

Confusion matrix:

True \ Predicted	🐼	🐻	🐻
🐼	14	0	0
🐻	0	9	1
🐻	1	2	10

Final evaluation of the model performance: test accuracy

Accuracy: $\frac{46}{50} = 92\%$

Error rate: $\frac{4}{50} = 8\%$

Confusion matrix:

True \ Predicted	🐼	🐻	🐻
🐼	**46**	0	0
🐻	1	**0**	0
🐻	3	0	**0**

Maybe this model will classify *everything* as a panda

Careful with imbalanced classes in the dataset!

The NN is a blackbox model

The parameterization of a NN is arbitrary and does not aim or attempt to reflect a process or a specific distribution

The weights are not directly interpretable parameters and the main parameters of interest are in the output layer

Check out our Bayesian neural network implementation!

https://github.com/dsilvestro/npBNN

Silvestro and Andermann 2020, arXiv: 2005.04987v1

Problems and misuses in AI