Model Learning

Matthew Engelhard

<u>*Learning* (or *training*) our model means</u>:

-> Finding specific values of our model parameters that predict y effectively from x in our training set

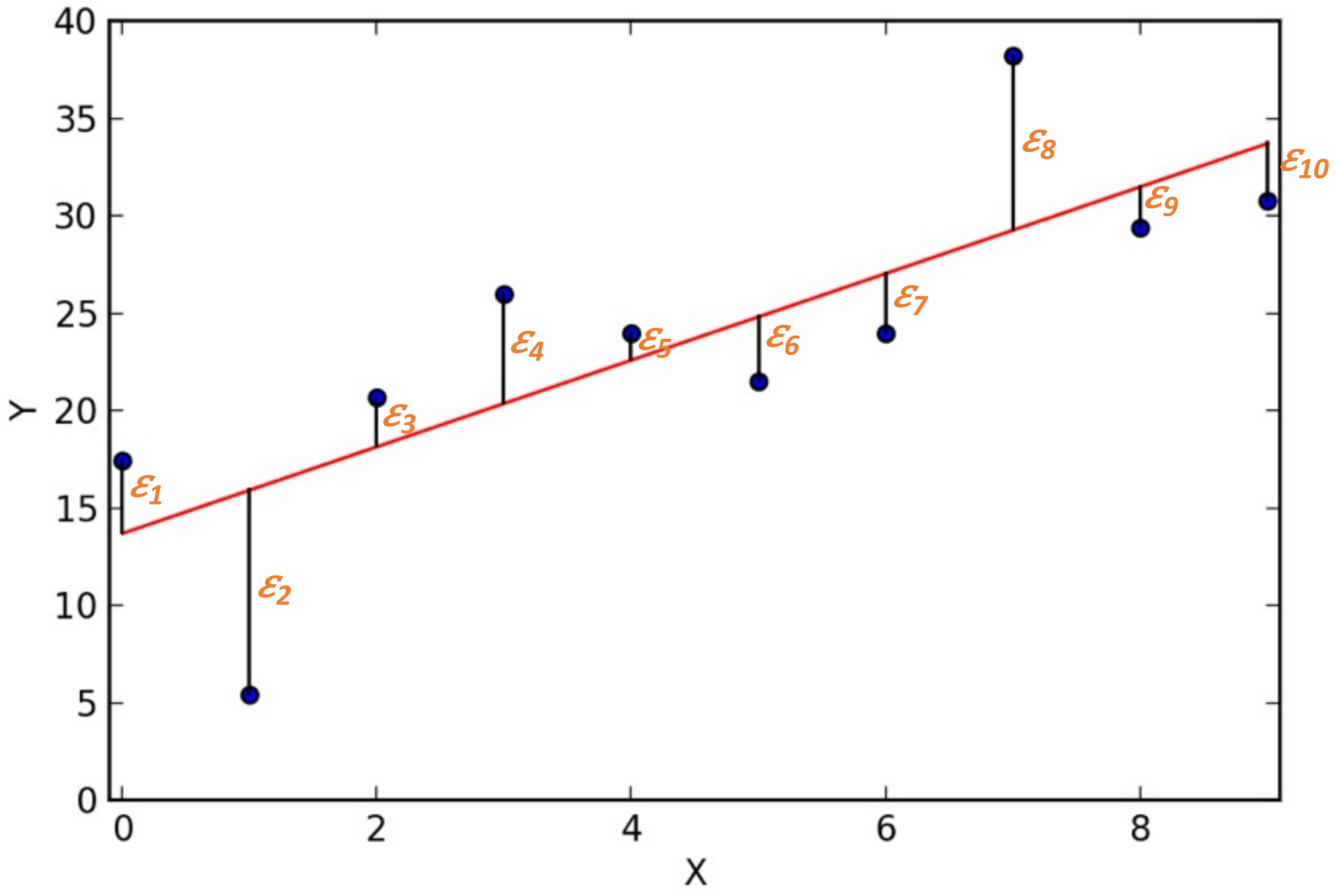
<u>What do we mean by "predict y effectively"?</u>

-> We need a number that measures how well we're predicting y

-> Then, we can set our parameters to the specific values that are best, according to this number

-> We call this number the "loss", and try to find parameters that minimize it

In *linear* regression, the loss is the mean squared error.



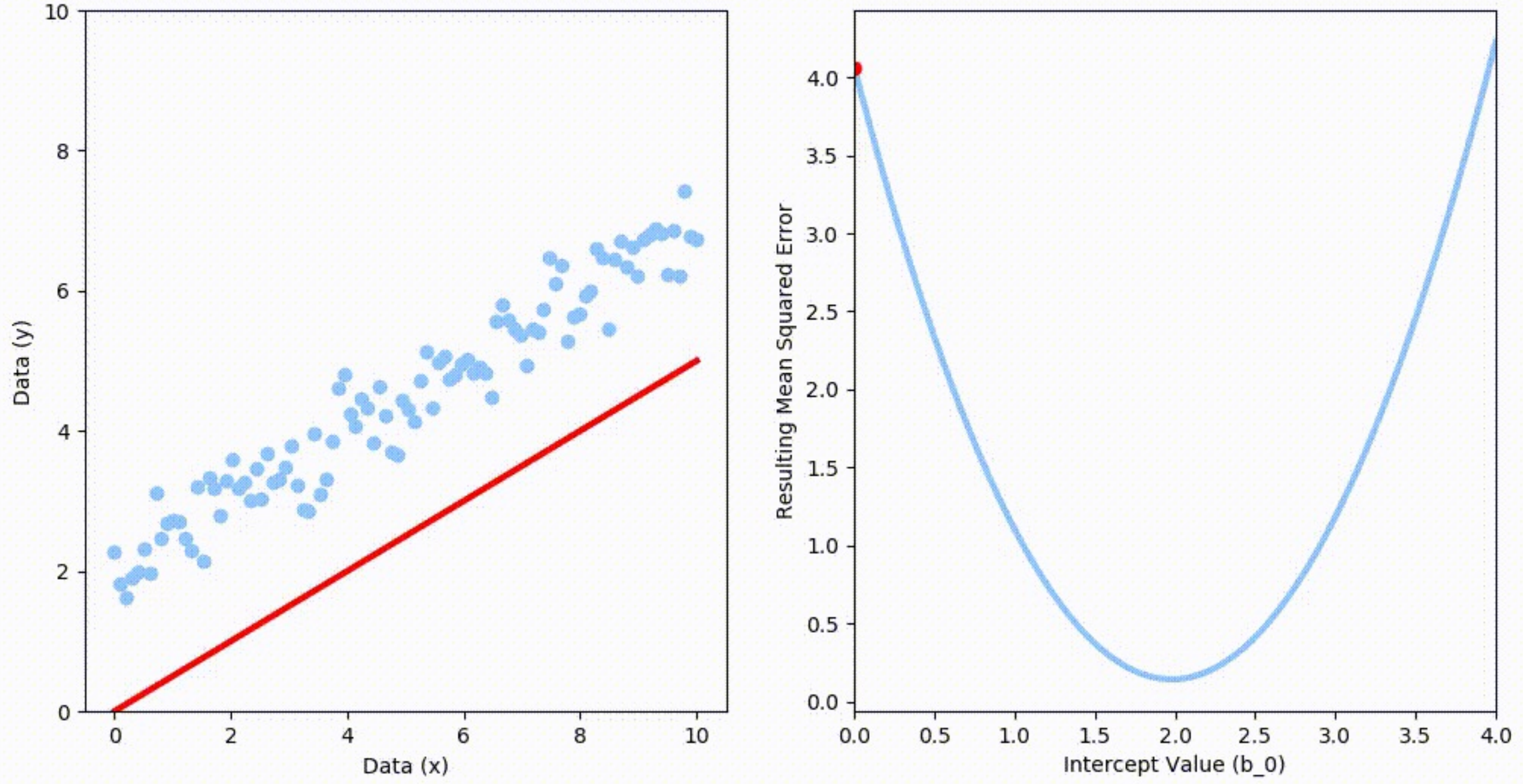
Error:

$$\varepsilon_i = y_i - \hat{y}_i$$

Mean square error (MSE)

$$\frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2$$

We have two parameters: the slope, and the intercept.
As we change the intercept, we can see that the MSE changes.
We're looking for values that minimize it.



What loss makes sense for *logistic* regression?

- Our model predicts the *probability* of death for each patient.
- If we change the parameters, we change the predicted probabilities for each patient.

$y_1 = 1$ (dies) $\quad$ $y_2 = 0$ (survives) $\quad$ $y_3 = 0$ (survives)

p_1? x p_2? x p_3?

What loss makes sense for *logistic* regression?

- Suppose we predict:
  - $p_1 = .1$
  - $p_2 = .9$
  - $p_3 = .7$

- Is this a good model? Why or why not?

$y_1 = 1$	$y_2 = 0$	$y_3 = 0$
(dies)	(survives)	(survives)

p_1? x p_2? x p_3?

What loss makes sense for *logistic* regression?

- Suppose we predict:
  - $p_1 = .8$
  - $p_2 = .3$
  - $p_3 = .1$
- Is this a good model? Why or why not?

$y_1 = 1$ (dies) | $y_2 = 0$ (survives) | $y_3 = 0$ (survives)

p_1? x p_2? x p_3?

What loss makes sense for *logistic* regression?

- Suppose we predict:
  - $p_1 = .8$
  - $p_2 = .3$
  - $p_3 = .1$

- What is the probability of the observed outcomes?

- This is called the likelihood. We want to find parameters that maximize it.

$y_1 = 1$ (dies) $\quad$ $y_2 = 0$ (survives) $\quad$ $y_3 = 0$ (survives)

p_1? x p_2? x p_3?

What loss makes sense for *logistic* regression?

- Suppose we predict:
  - $p_1 = .8$
  - $p_2 = .3$
  - $p_3 = .1$

- Is this a good model? Why or why not?

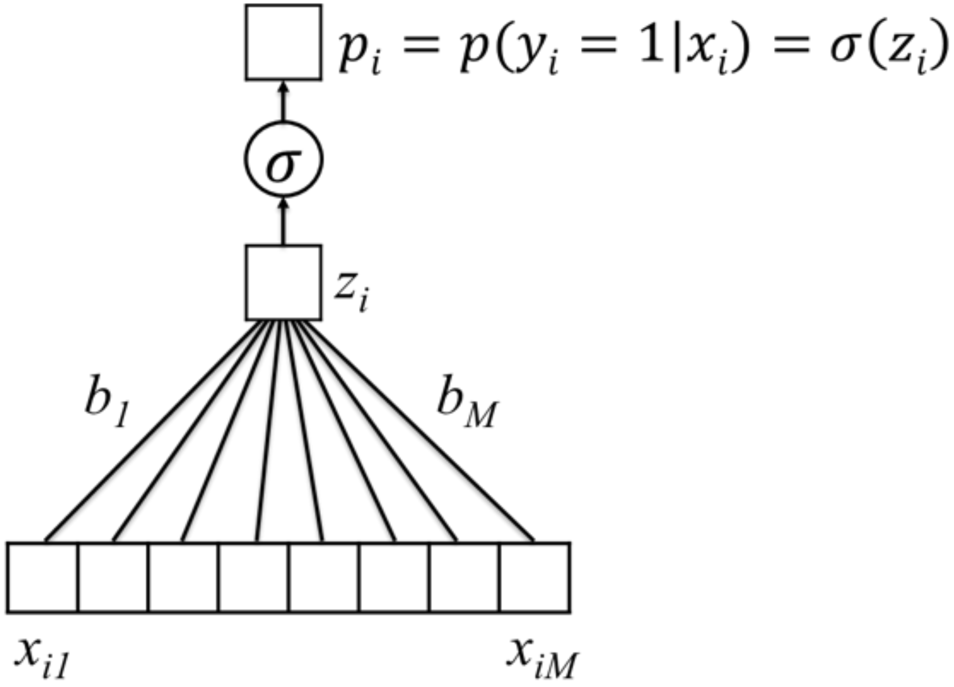
- Our parameters affect *all* the predictions: changing a parameter to *decrease* y_2 may also *increase* y_3

$y_1 = 1$ (dies) $y_2 = 0$ (survives) $y_3 = 0$ (survives)

p_1? x p_2? x p_3?

Logistic Regression Likelihood

Probability of all the events $y_1, \dots, y_N$ given our current model parameters

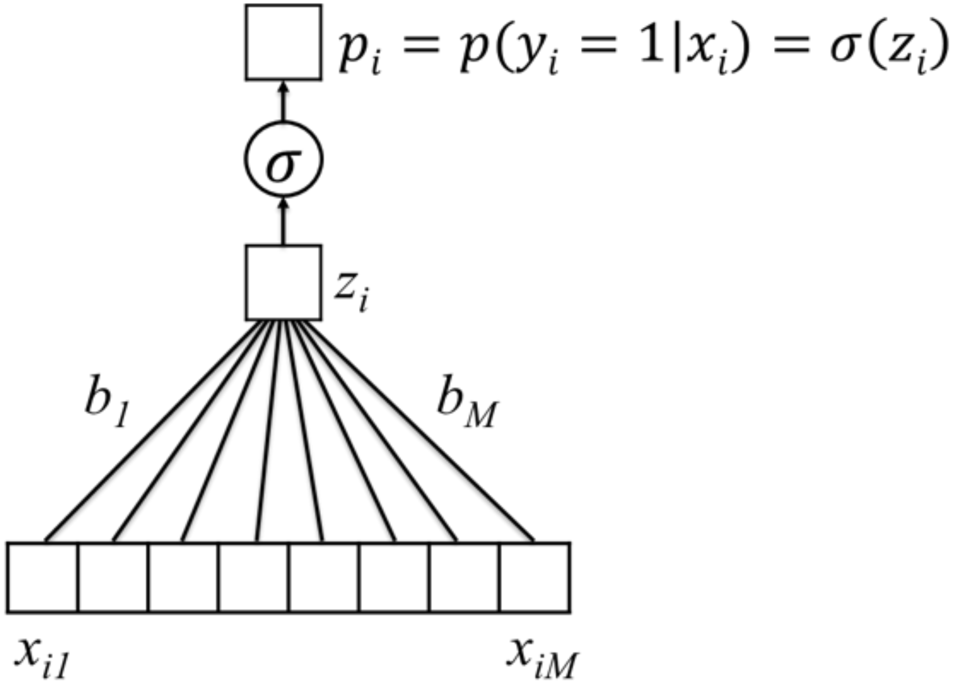


The predicted probability of y_i is:

$$\begin{cases} p_i & \text{if } \; y_i = 1 \\ 1 - p_i & \text{if } \; y_i = 0 \end{cases}$$

Logistic Regression Likelihood

Probability of all the events $y_1, \dots, y_N$ given our current model parameters



The predicted probability of y_i is:

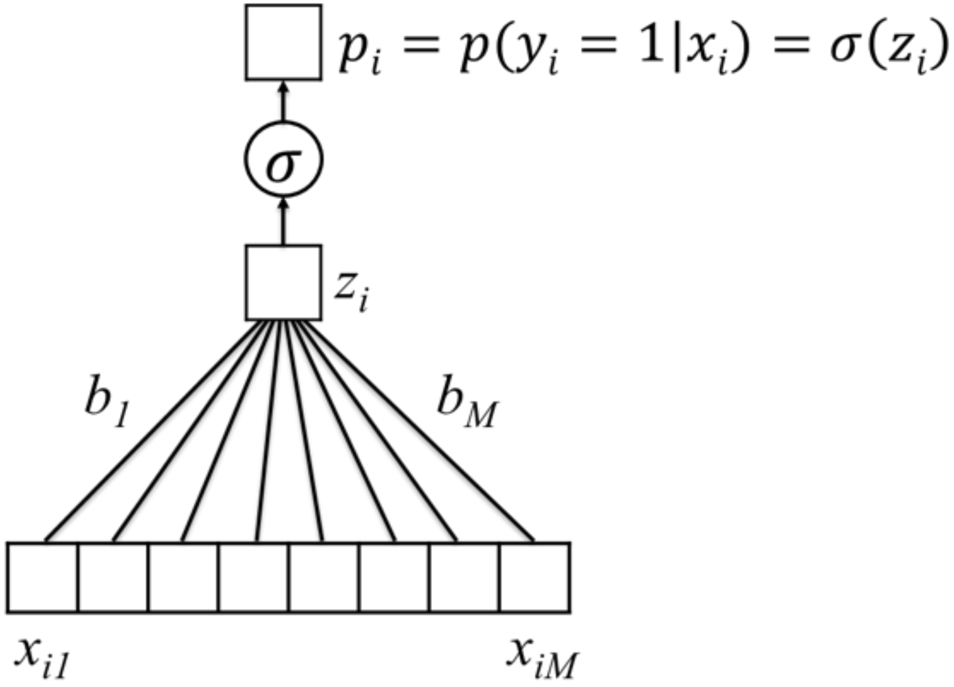
$$\begin{cases} p_i & \text{if } \; y_i = 1 \\ 1 - p_i & \text{if } \; y_i = 0 \end{cases}$$

Is there a way to write this without the *if* statement? Try the following:

$$p_i^{y_i}(1 - p_i)^{1-y_i}$$

Logistic Regression Likelihood

Probability of all the events $y_1, \ldots, y_N$ given our current model parameters



The predicted probability of y_i is:

$$\begin{cases} p_i & \text{if } \; y_i = 1 \\ 1 - p_i & \text{if } \; y_i = 0 \end{cases}$$

Is there a way to write this without the *if* statement? Try the following:

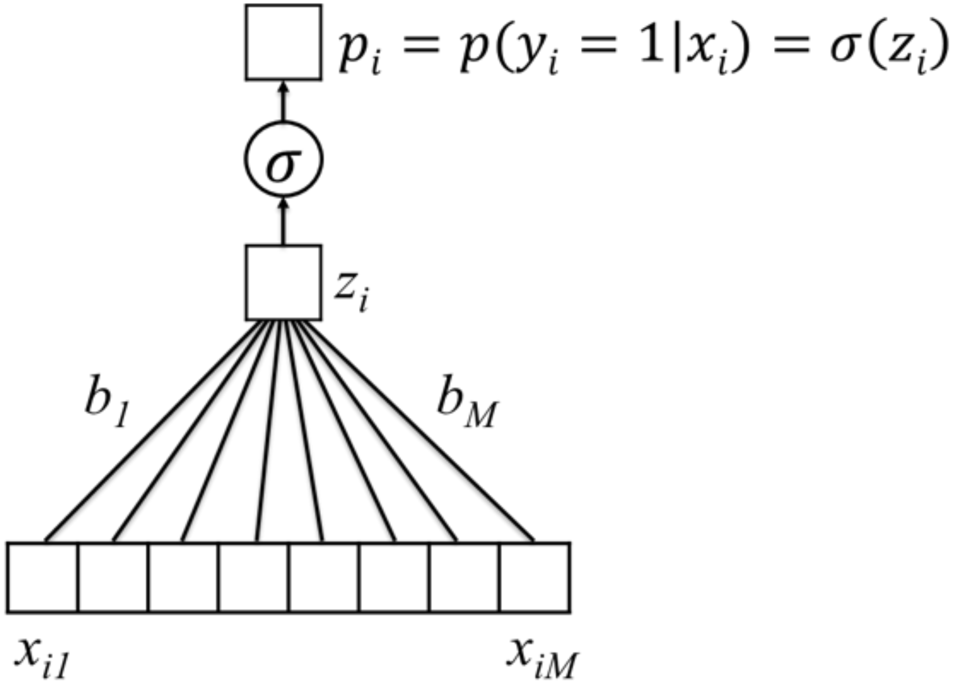
$$p_i^{y_i}(1 - p_i)^{1-y_i}$$

Now we multiply all the predicted probabilities together:

$$\prod_{i=1}^{N} (p_i)^{y_i}(1 - p_i)^{1-y_i}$$

Logistic Regression Likelihood

Probability of all the events $y_1, \dots, y_N$ given our current model parameters



Now we multiply all the predicted probabilities together:

$$\prod_{i=1}^{N} (p_i)^{y_i}(1 - p_i)^{1-y_i}$$

Often we see $\sigma(z_i)$ substituted for p_i:

$$\prod_{i=1}^{N} \sigma(z_i)^{y_i}(1 - \sigma(z_i))^{1-y_i}$$

Two Final Modifications

Again, here's the *likelihood*:

$$\prod_{i=1}^{N} \sigma(z_i)^{y_i}(1-\sigma(z_i))^{1-y_i}$$

Two Final Modifications

Again, here's the *likelihood*:

$$\prod_{i=1}^{N} \sigma(z_i)^{y_i}(1 - \sigma(z_i))^{1-y_i}$$

1. For numerical stability, we instead work with the *log-likelihood*:

$$\sum_{i=1}^{N} y_i \log \sigma(z_i) + (1 - y_i) \log\big(1 - \sigma(z_i)\big)$$

Two Final Modifications

Again, here's the *likelihood*:

$$\prod_{i=1}^{N} \sigma(z_i)^{y_i} (1 - \sigma(z_i))^{1-y_i}$$

1. For numerical stability, we instead work with the *log-likelihood*:

$$\sum_{i=1}^{N} y_i \log \sigma(z_i) + (1 - y_i) \log\big(1 - \sigma(z_i)\big)$$

2. And by convention / for consistency, we minimize the *negative* log-likelihood rather than maximizing the positive:

$$-\sum_{i=1}^{N} y_i \log \sigma(z_i) + (1 - y_i) \log\big(1 - \sigma(z_i)\big)$$

Cross-Entropy Loss

Again, here's the *likelihood*:

$$\prod_{i=1}^{N} \sigma(z_i)^{y_i}(1-\sigma(z_i))^{1-y_i}$$

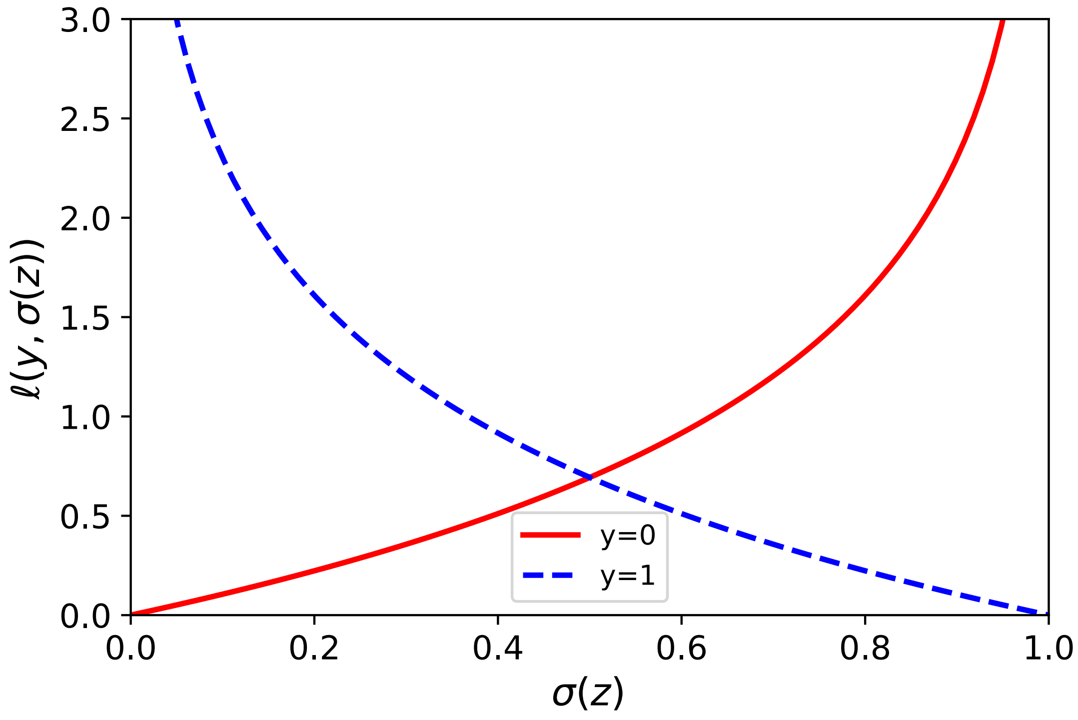
1. For numerical stability, we instead work with the *log-likelihood*:

$$\sum_{i=1}^{N} y_i \log \sigma(z_i) + (1-y_i)\log\big(1-\sigma(z_i)\big)$$

2. And by convention / for consistency, we minimize the *negative* log-likelihood rather than maximizing the positive:

$$-\sum_{i=1}^{N} y_i \log \sigma(z_i) + (1-y_i)\log\big(1-\sigma(z_i)\big)$$

This is called the *cross-entropy loss.*
We are looking for parameters that make it as small as possible.
It is used for *all* the prediction tasks we consider in this course.



How do we minimize the loss?

- The cross-entropy loss just tells us *what* quantity we should be minimizing
- In some cases (e.g. linear regression), we can solve for the minimum directly
- But, we’d like to have an approach that works even for very complex models

Strategy: determine how small changes in parameters affect the loss

MORTALITY PREDICTION WORKSHEET

COVARIATES

patient	age	age_normalized	female	temp	temp_normalized
0	30.5	-0.5	0	105.0	2.4
1	74.0	1.1	1	96.7	-0.8
2	27.4	-0.6	0	96.1	-1.0
3	0.1	-1.5	1	98.5	-0.1
4	0.7	-1.5	1	96.5	-0.9
5	49.9	0.2	1	97.1	-0.6
6	72.9	1.0	1	100.1	0.5
7	29.1	-0.5	1	99.6	0.3
8	83.5	1.4	1	100.6	0.7
9	82.3	1.4	1	95.2	-1.3
10	23.7	-0.7	0	99.4	0.2
11	12.9	-1.1	0	96.6	-0.8
12	53.9	0.4	1	100.3	0.6
13	18.8	-0.9	0	98.6	0.0
14	51.8	0.3	0	98.5	-0.1
15	3.3	-1.4	0	94.6	-1.6
16	69.7	0.9	0	99.1	0.1
17	60.4	0.6	1	104.2	2.1
18	73.6	1.1	1	99.1	0.1
19	53.3	0.3	1	99.1	0.1

OUTCOMES AND PREDICTIONS

mortality	predicted_log_odds	predicted_prob	prediction	correct?	loss
1	0.00	0.50	0	0	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
1	0.00	0.50	0	0	0.3010
0	0.00	0.50	0	1	0.3010
1	0.00	0.50	0	0	0.3010
1	0.00	0.50	0	0	0.3010
1	0.00	0.50	0	0	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
0	0.00	0.50	0	1	0.3010
1	0.00	0.50	0	0	0.3010
1	0.00	0.50	0	0	0.3010
0	0.00	0.50	0	1	0.3010

PARAMETERS

	b_age	b_female	b_temp	bias
guess	0.00	0.00	0.00	0.00
optimal	[redacted]			

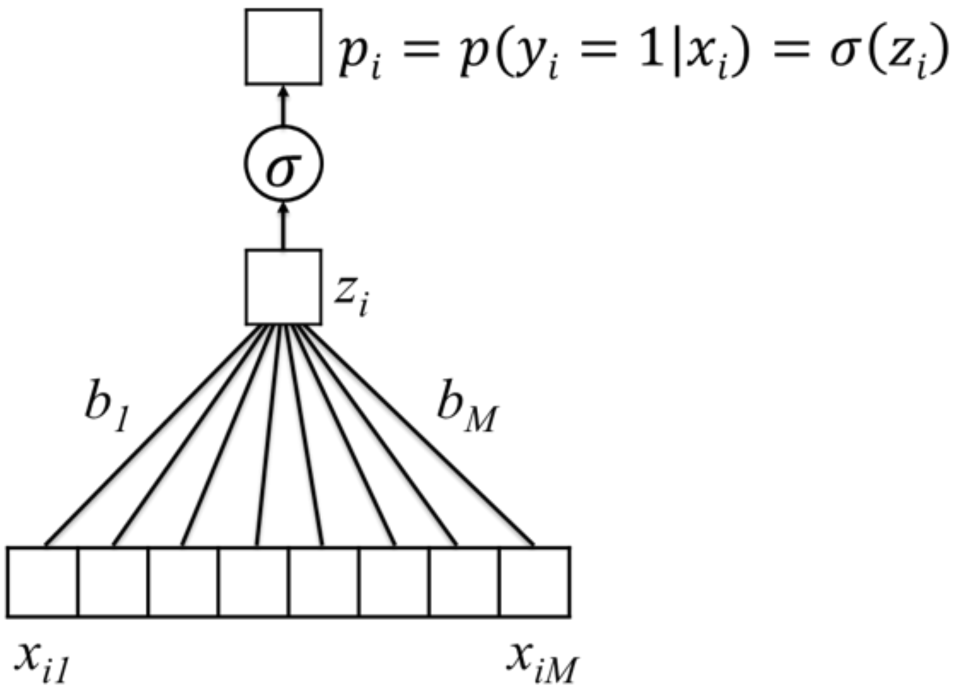
PERFORMANCE

accuracy	avg_loss
0.65	0.3010

How does changing a parameter change the loss?

If we change any parameter – let's say b_1 – it will change:

- z_i (for each patient i)
- which in turn affects p_i (for each patient i)
- Which affects the loss



A chain of effects…

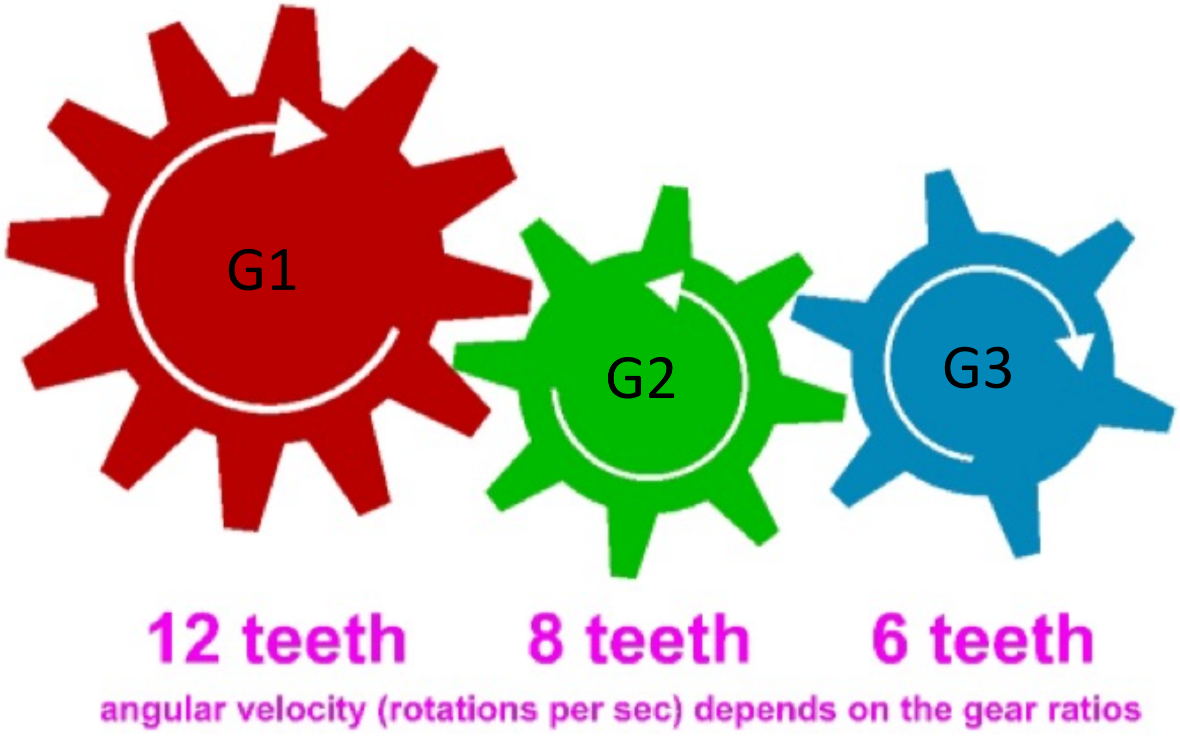
<u>We know:</u>

- If we rotate G1 by 1 radian, G2 will rotate by –12/8 radians.
- If G2 rotates by 1 radian, G3 will rotate by -8/6 radians.

How do we determine the effect of G1 on G3?

➢ Multiply the effects.

➢ $\left(-\frac{12}{8}\right) * \left(-\frac{8}{6}\right) = \frac{12}{6} = 2$



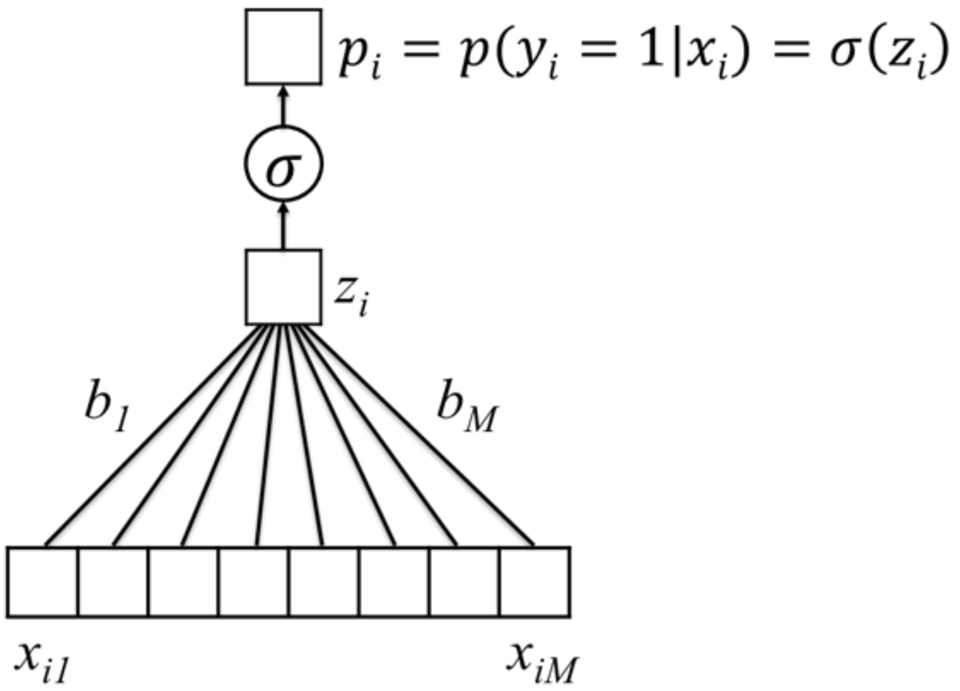
A chain of effects…

<u>We know:</u>

- If we increase b_1 by a small amount ε, then z_i will increase by $\varepsilon * x_{i1}$
- If we increase z_i by a small amount ε, then p_i will increase by $\varepsilon * \frac{d\sigma(z_i)}{dz_i}$ (depends on z_i)

How do we determine the effect b_1 on the cross-entropy loss (which depends on p_i)?

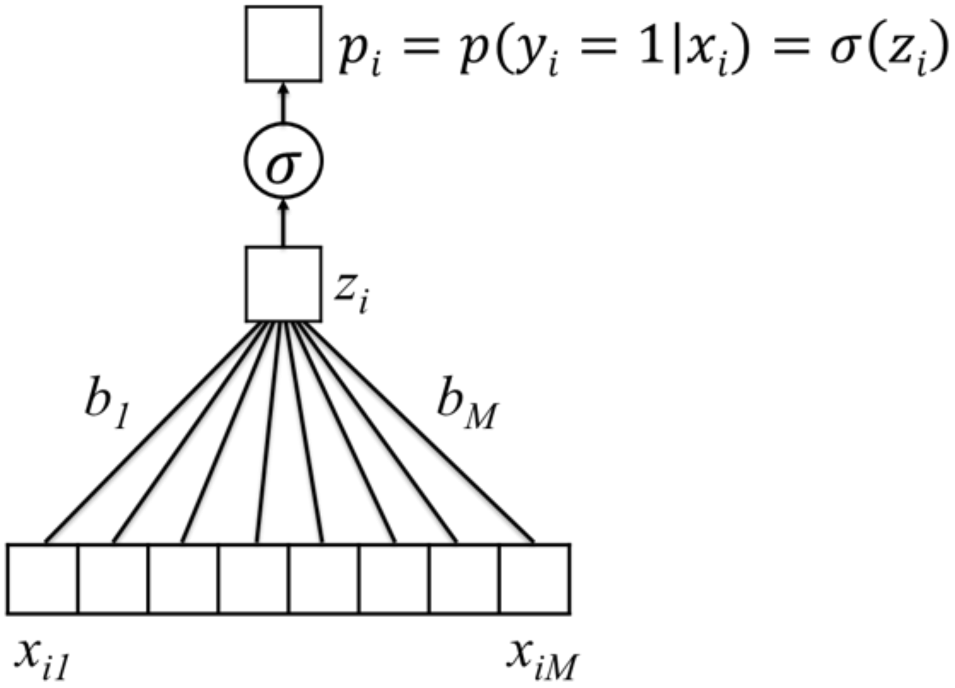
➢ Multiply the effects.



Our strategy: use principles of calculus – slope/gradient – to determine how small changes in parameters affect the loss

<u>We use the *chain rule* (calc 101) to account for the chain of effects.</u>

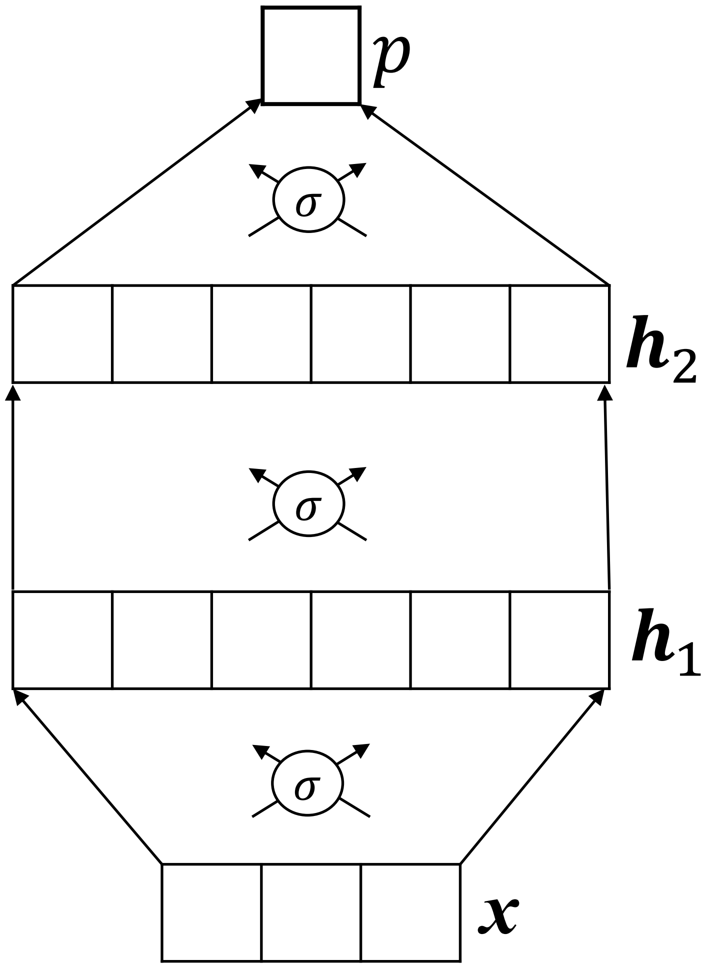
- Could be a very long chain…
- Some parameters have a greater effect than others
- We change all parameters at once, with each change proportional to that parameter's effect on the loss
  - This is *gradient descent*



Our strategy: use principles of calculus – slope/gradient – to determine how small changes in parameters affect the loss

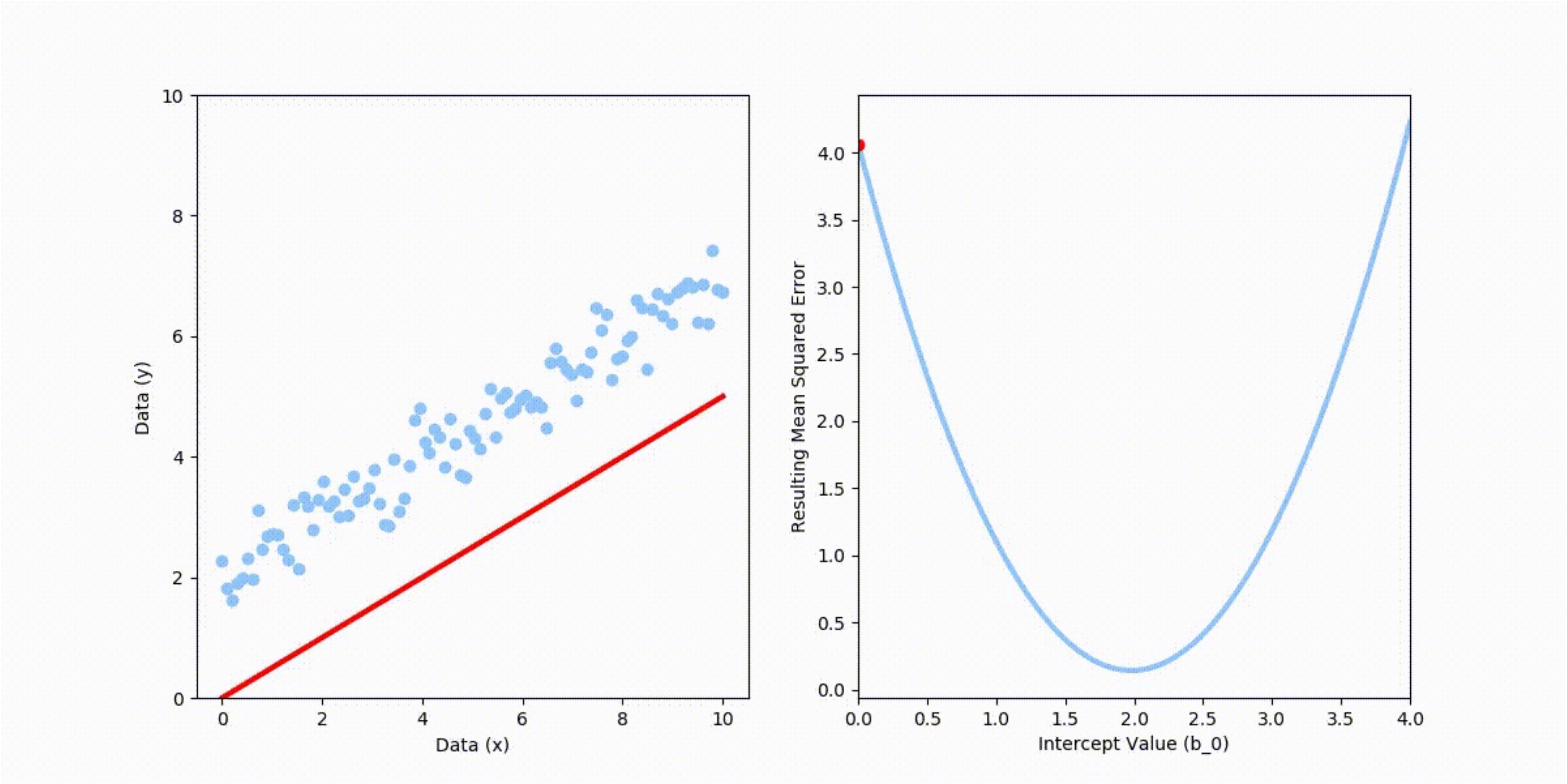
<u>It could be a very long (and complex) chain…</u>

- If we increase x_1 by ε, it changes *all* of the h_{1j}…
- …each of which changes *all* of the h_{2j}
- …each of which changes p
- Machine learning software like TensorFlow allows us to keep track, even for very complicated models



For simple models, minimizing the loss is easy.

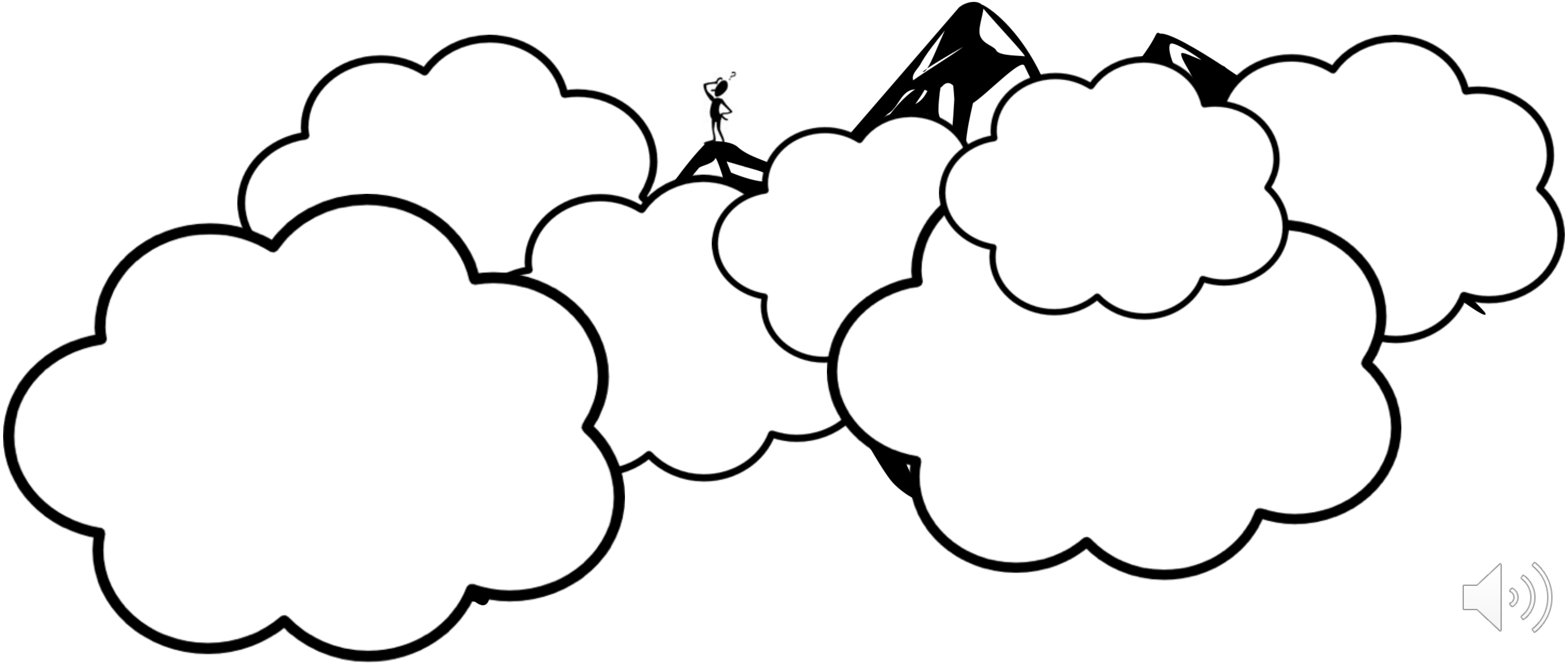
1. There are a limited number of parameters to consider
2. Here, the loss is ‘bowl-shaped’, or *convex*; we can simply walk downhill from our current position
3. For linear regression, we can simply *solve* for the best parameters; but in general this is not true



For complex models, it is much more difficult…

1. High-dimensional
2. Non-convex
3. No closed-form solution

…and we're never sure we've found the *best* parameters



Learning in Neural Networks

- With deep learning models, we are trying to minimize a function of many variables
- Can’t visualize it
- Can’t solve for the minimum directly
- So, we follow the slope and hope for the best (i.e. gradient descent)
- May end up in a low point that isn’t the lowest, i.e. *local minimum*
- But, if we have lots of data, things usually work out OK

Conclusions

- *Learning* consists in setting model parameters to maximize some measure of fit; or equivalently, minimize some loss

- In classification problems, the loss we wish to minimize is called the cross-entropy loss, and it related to the probability of the observed outcomes supposing our current model were correct

- *Backpropagation* allows us to determine how the loss is affected by small changes in each model parameter. We can apply this information repeatedly to reduce the loss.

- In deep learning, we can never be sure that we’ve found the best possible parameters, but in practice, we find good parameter values that generalize well