Neural networks

STAT 471

November 23, 2021

Where we are

- ✓ **Unit 1:** Intro to modern data mining
- ✓ **Unit 2:** Tuning predictive models
- ✓ **Unit 3:** Regression-based methods
- ✓ **Unit 4:** Tree-based methods
- **Unit 5:** Deep learning

Lecture 1: Deep learning preliminaries

Lecture 2: Neural networks

Lecture 3: Deep learning for images

Lecture 4: Deep learning for text

Lecture 5: Unit review and quiz in class

Homework 5 due the following Wednesday.

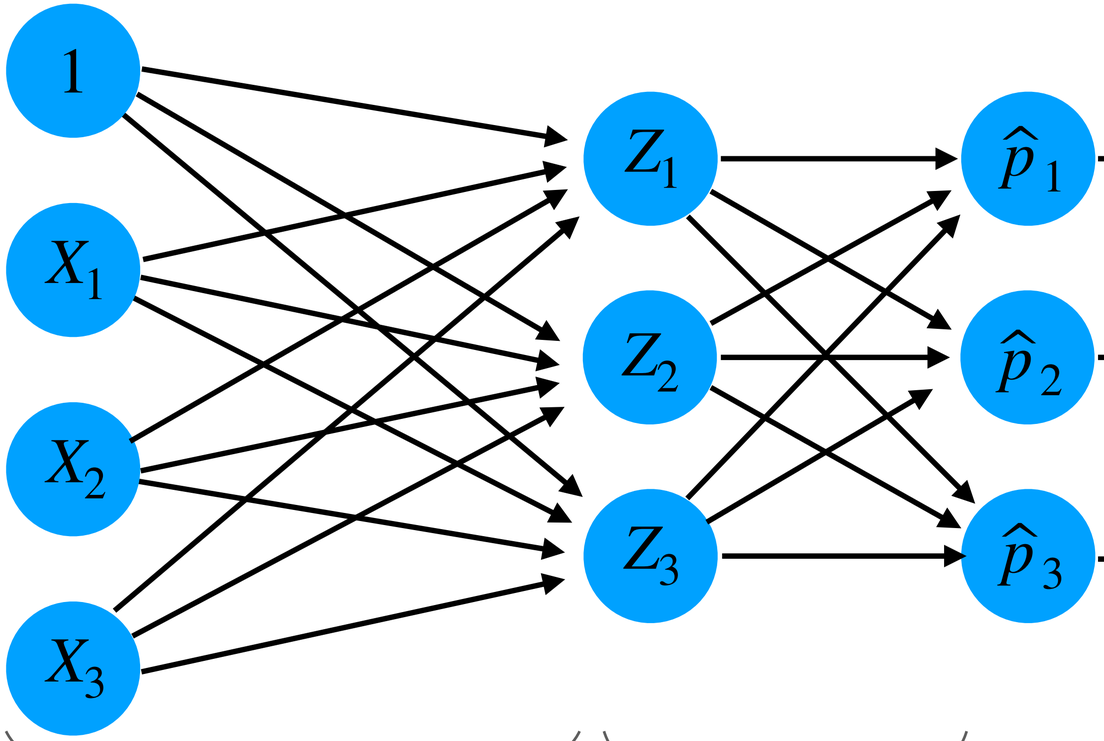
Recall: Multi-class logistic model

score for how "x" like it is

Input

Input

Input



0.95

0.04

0.01

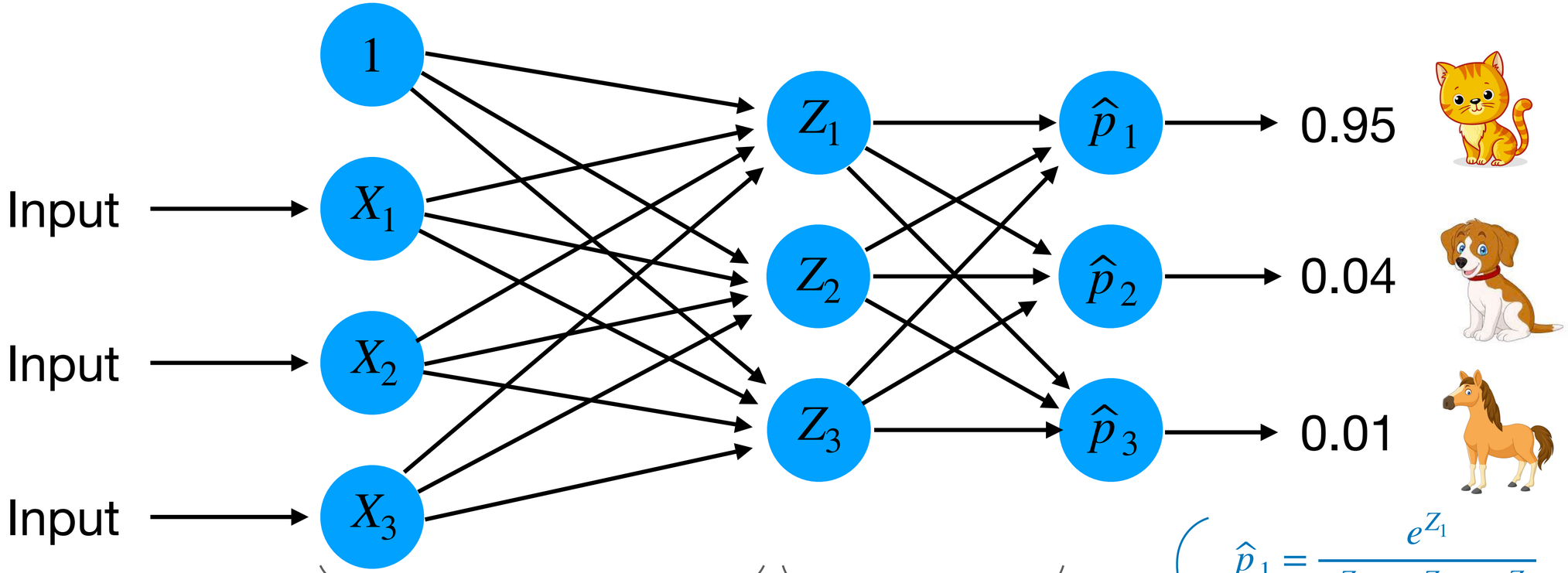
summation softmax activation

$$\hat{p}_1 = \frac{e^{Z_1}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}$$

$$\hat{p}_2 = \frac{e^{Z_2}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}$$

$$\hat{p}_3 = \frac{e^{Z_3}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}$$

Recall: Multi-class logistic model



$$\hat{p}_1 = \frac{e^{Z_1}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}$$

$$\hat{p}_2 = \frac{e^{Z_2}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}$$

$$\hat{p}_3 = \frac{e^{Z_3}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}$$

layers are able to pick up on higher level features

This simple architecture directly uses raw pixel features.
Intuitively, a good classifier would pick up on cat's eyes, ears, etc.

One-hidden-layer neural network

Input layer

Hidden layer

Output layer

1, X_1, X_2, X_3

b_1, b_2, w_1, w_2, w_3, w_4, w_5, w_6

Z_{11}, Z_{12}

1, A_{11}, A_{12}

b_3, b_4, b_5, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}

Z_{21}, Z_{22}, Z_{23}

$\hat{p}_1$, $\hat{p}_2$, $\hat{p}_3$

0.95

0.04

0.01

summation

ReLU activation

summation

softmax activation

add in one layer to picture

One-hidden-layer neural network

Input layer

Hidden layer

Output layer

layer combination

applying features

1, X_1, X_2, X_3

b_1, b_2, w_1, w_2, w_3, w_4, w_5, w_6

Z_{11}, Z_{12}

1, A_{11}, A_{12}

b_3, b_4, b_5, w_7, w_8, w_9, w_{10}, w_{11}, w_{12}

Z_{21}, Z_{22}, Z_{23}

$\hat{p}_1$, $\hat{p}_2$, $\hat{p}_3$

0.95

0.04

0.01

summation

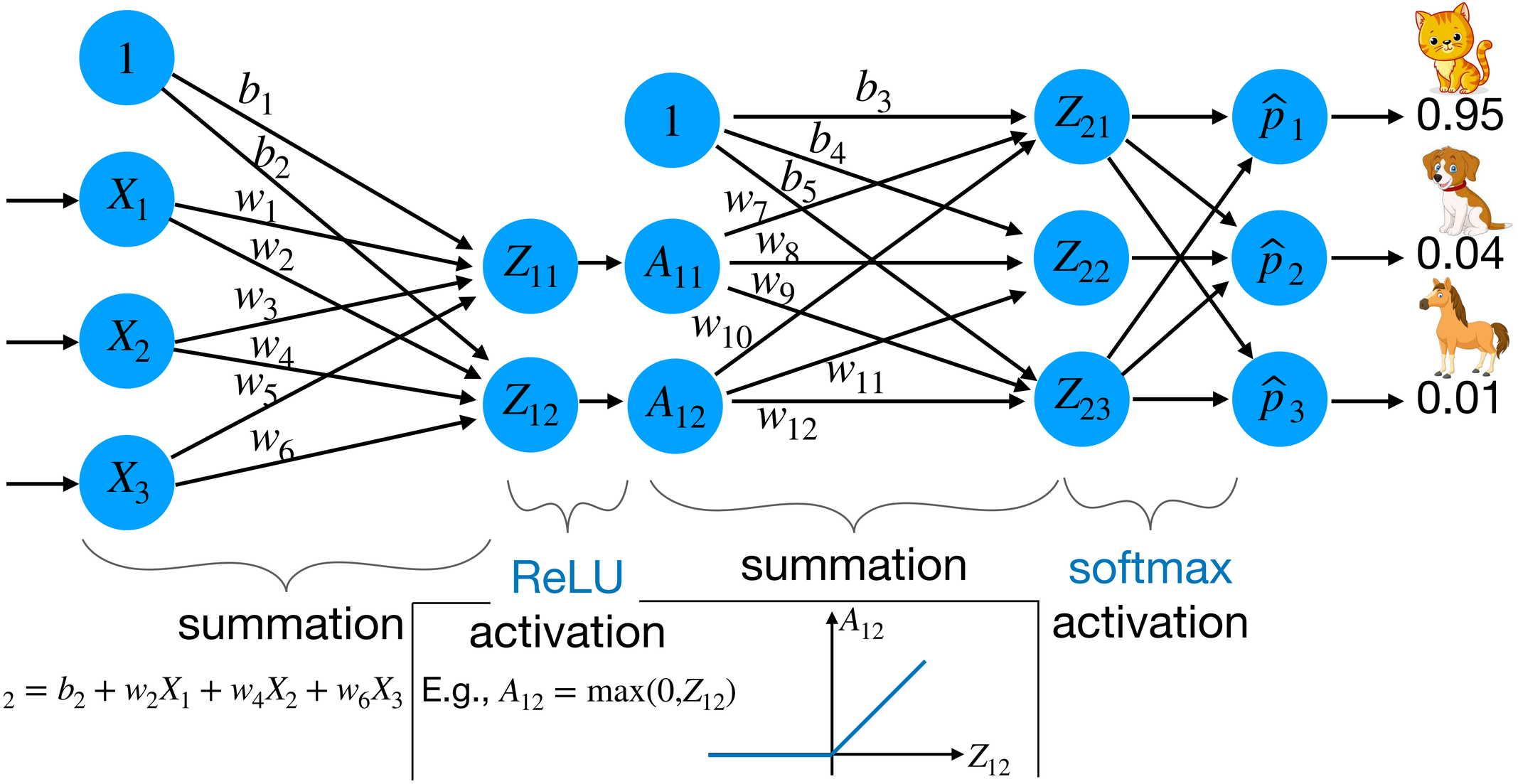
ReLU activation

summation

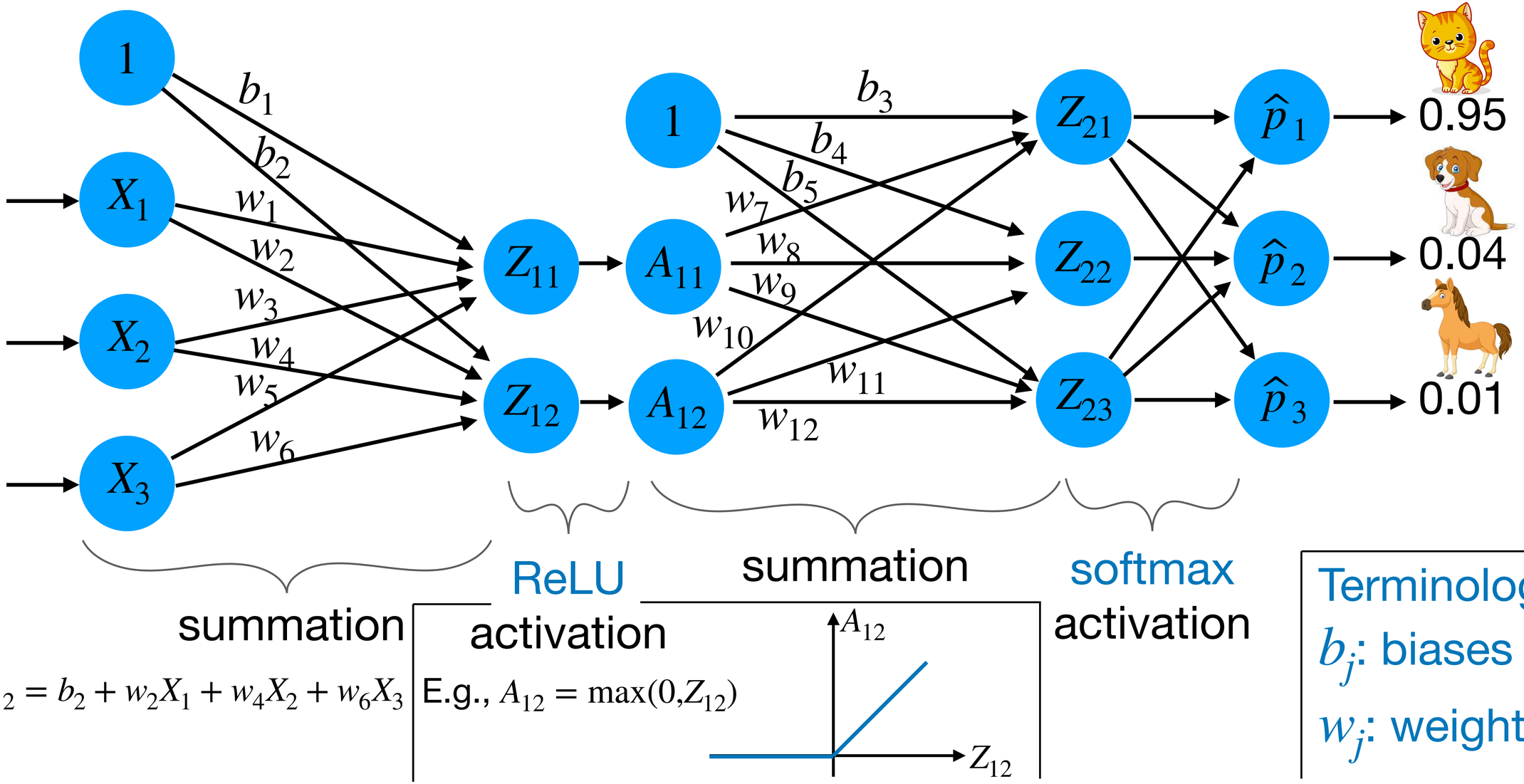
softmax activation

E.g., $Z_{12} = b_2 + w_2 X_1 + w_4 X_2 + w_6 X_3$

One-hidden-layer neural network



One-hidden-layer neural network



Terminology:

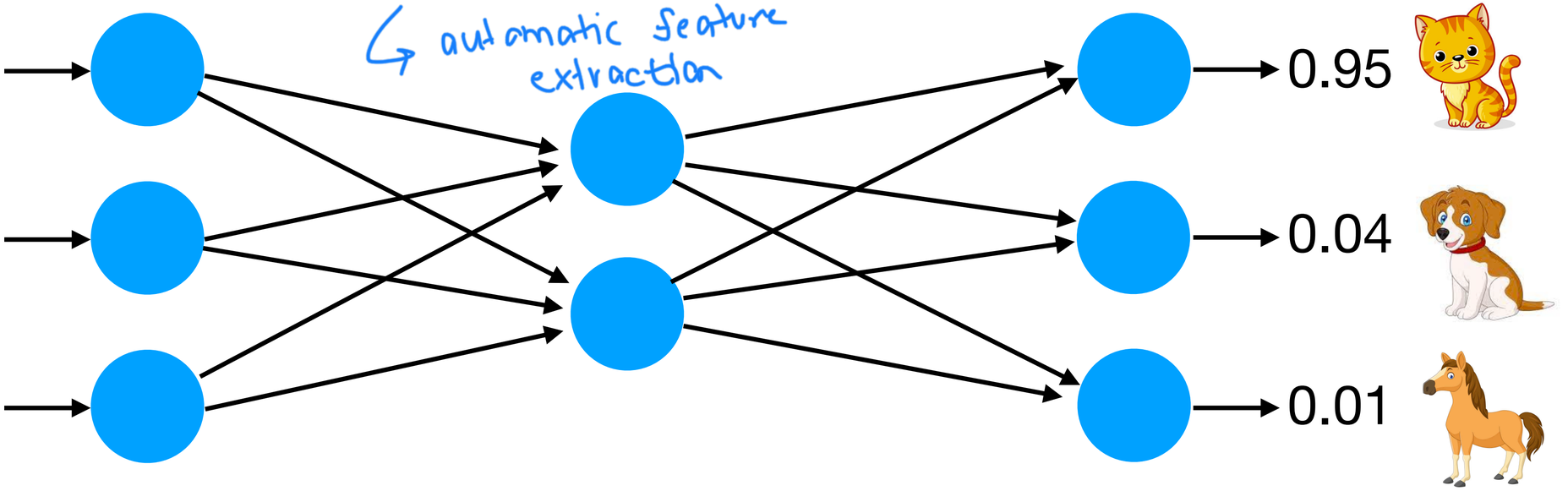
b_j: biases

w_j: weights

One-hidden-layer neural network

Concise representation

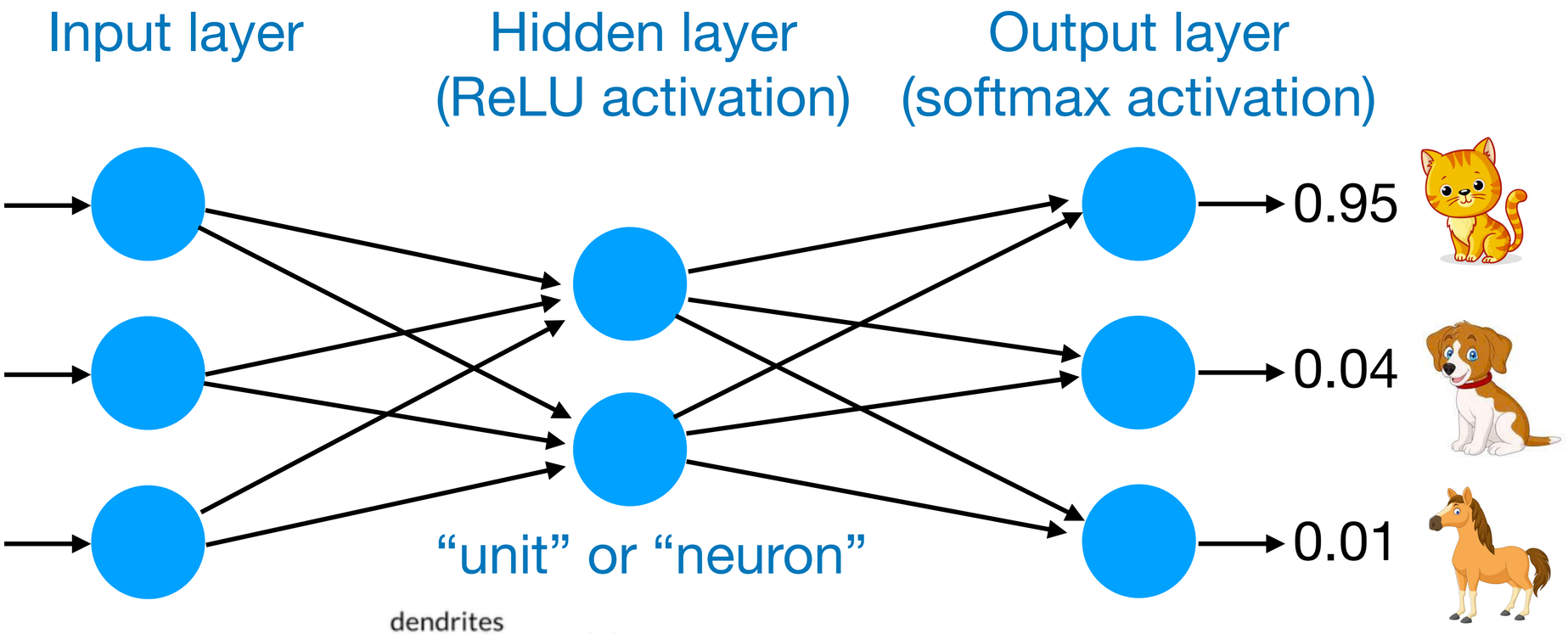
try to make it similar to how brain operates

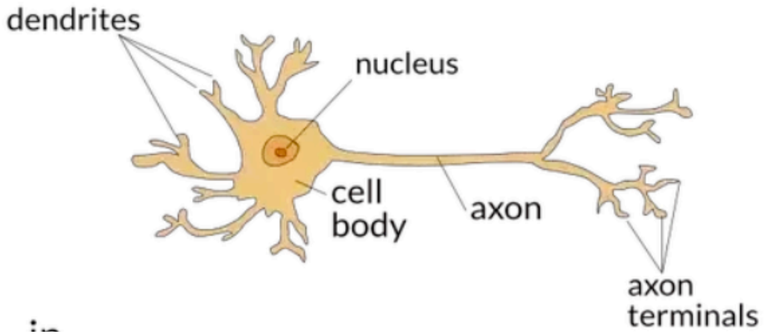


bias terms still floating around, but just not pictured

One-hidden-layer neural network

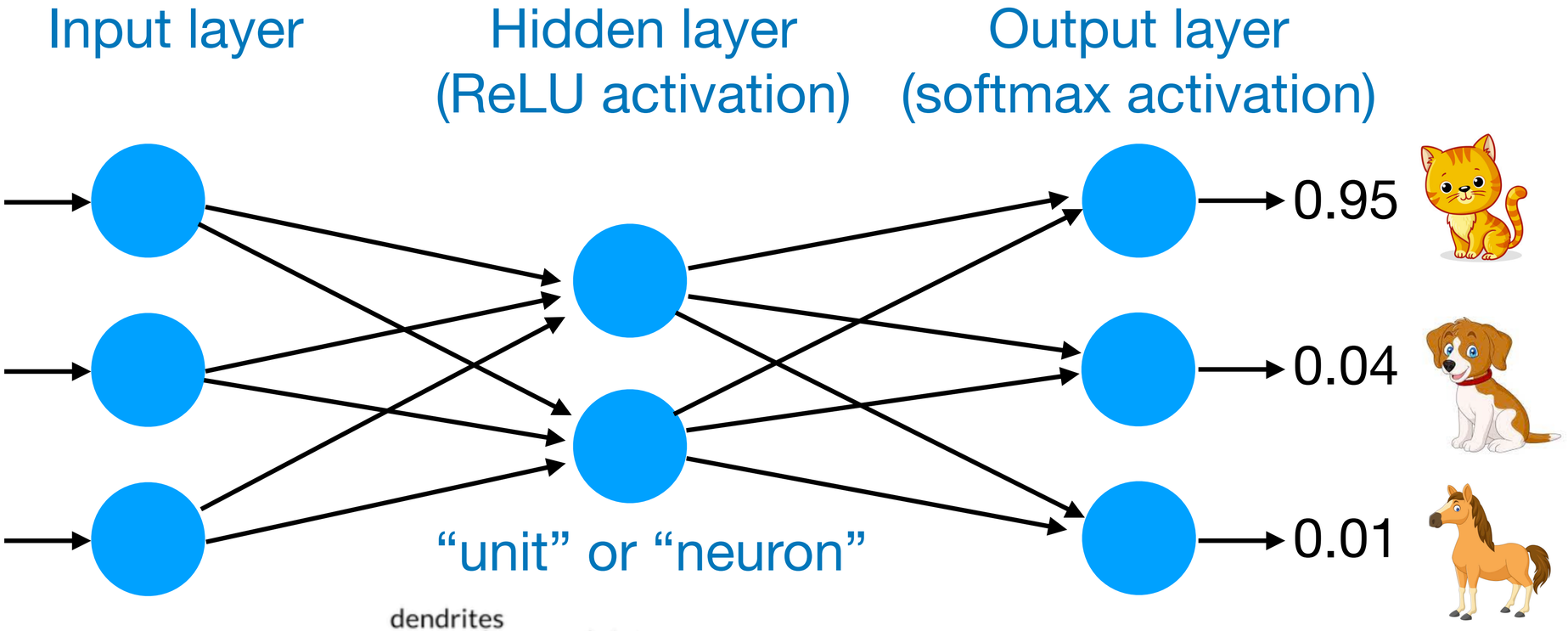
Concise representation



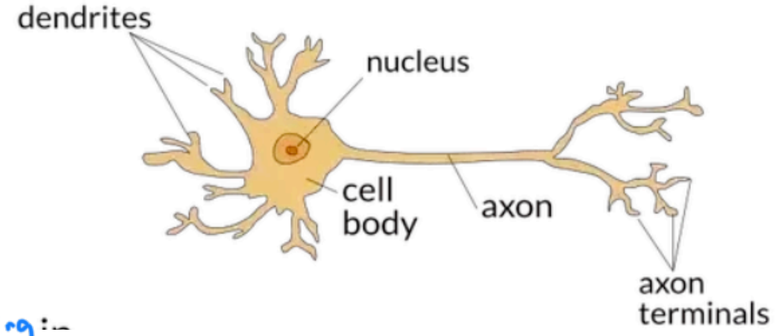


One-hidden-layer neural network

Concise representation



learns what parts of image should be paying attention to
↳ but could have no idea what seeing

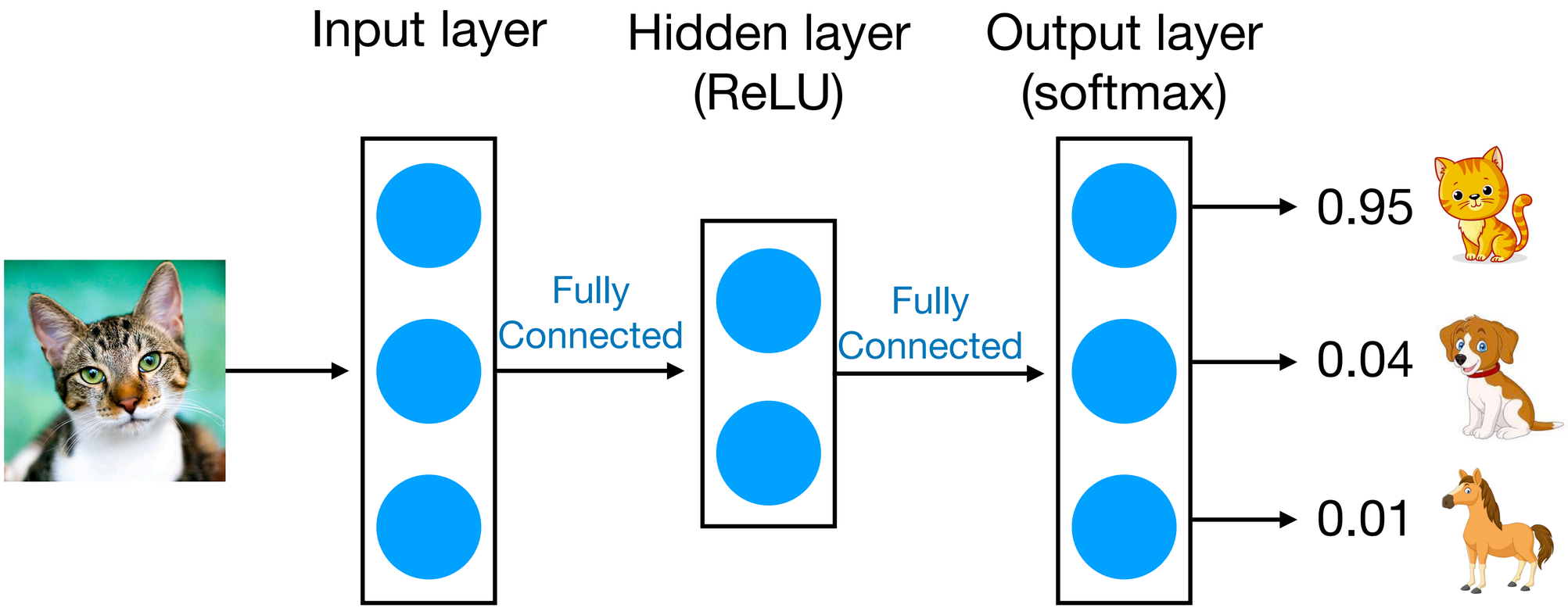


Notes:

- Bias terms left implicit.
- Fully connected architecture.
- Hidden layer: learned features.

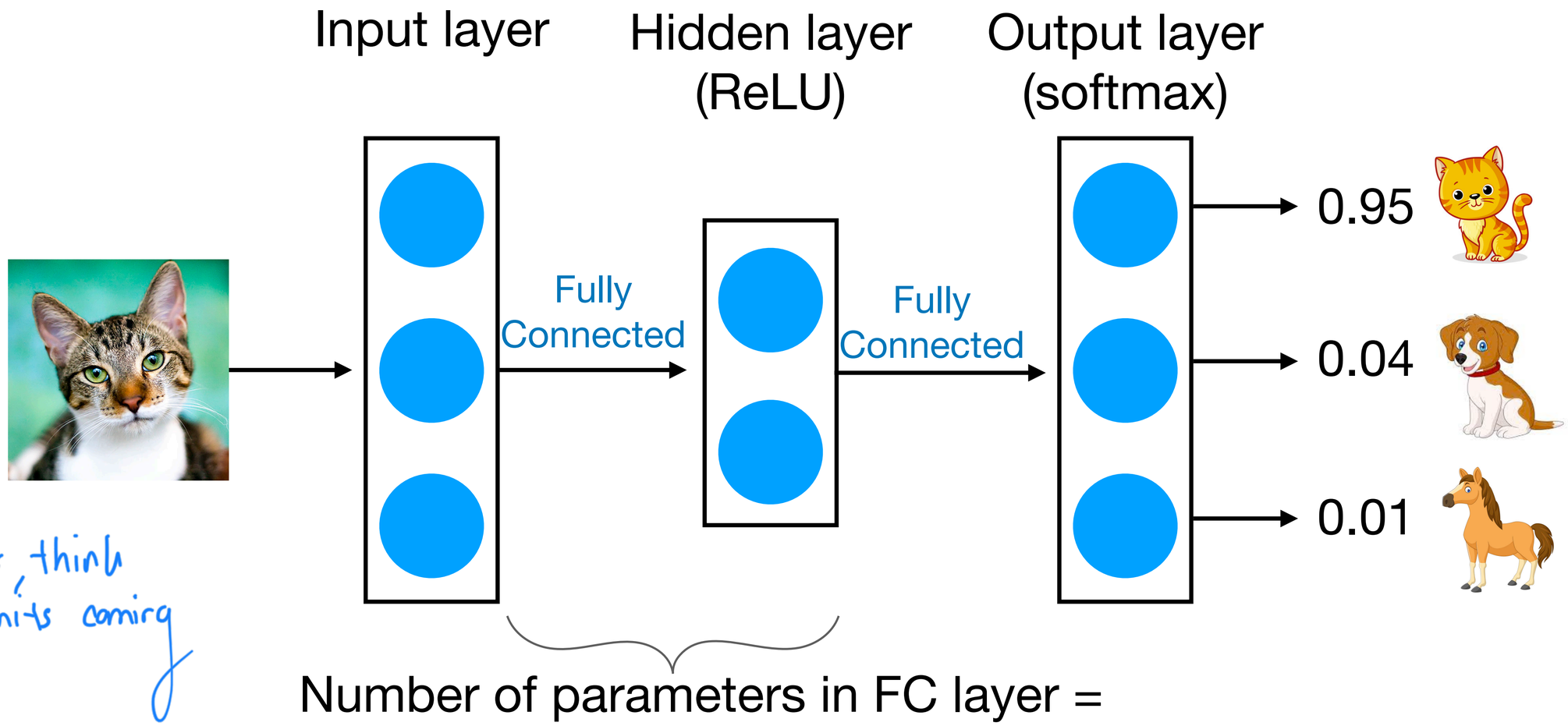
One-hidden-layer neural network

Even more concise representation



One-hidden-layer neural network

Even more concise representation



for each layer, think about # units coming into it

Number of parameters in FC layer =

(# units in previous layer + 1) × (# units in this layer)

Multi-layer fully connected neural networks

how you get term deep learning

ReLU is nonlinear, which allows it to build up complexity

Input layer | Hidden layer 1 (ReLU) | Hidden layer 2 (ReLU) | Output layer (softmax)

Fully Connected

Fully Connected

Fully Connected

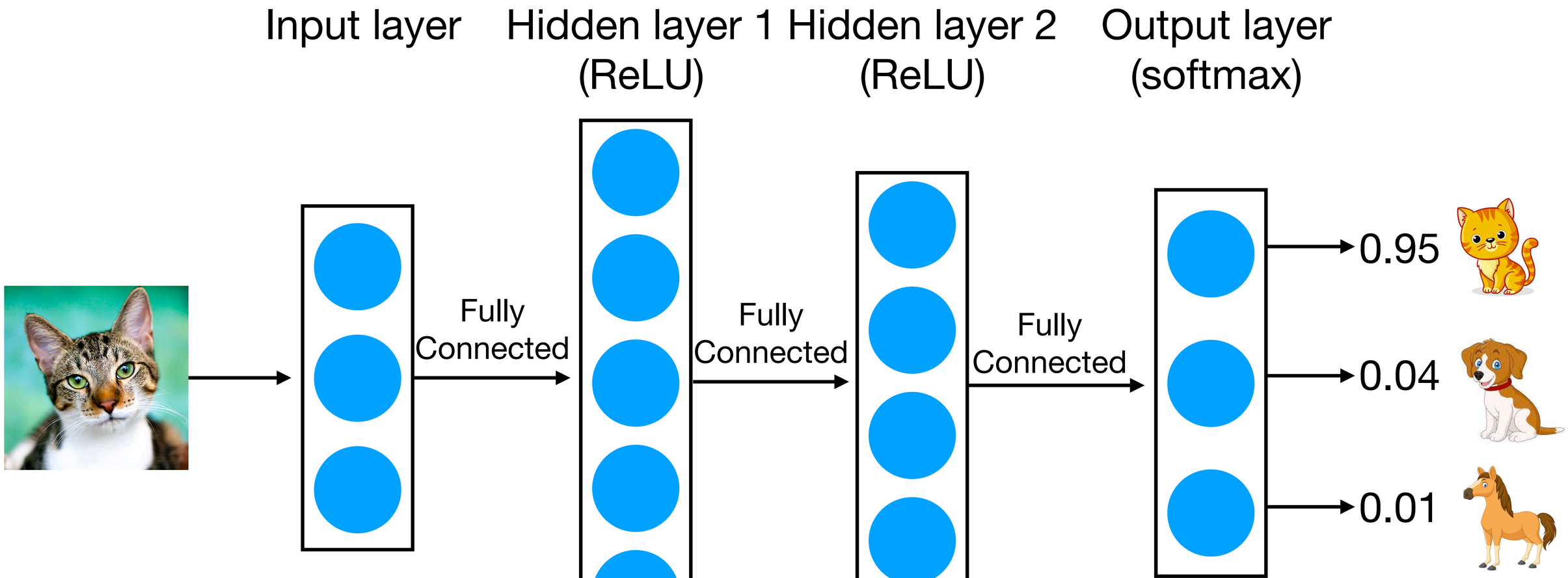
0.95

0.04

0.01

start w/ raw representation & get more abstract representations

Multi-layer fully connected neural networks



Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

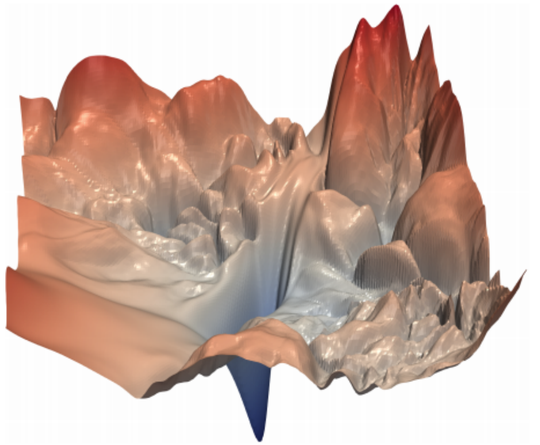
Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.

Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$



https://arxiv.org/abs/1712.09913

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.

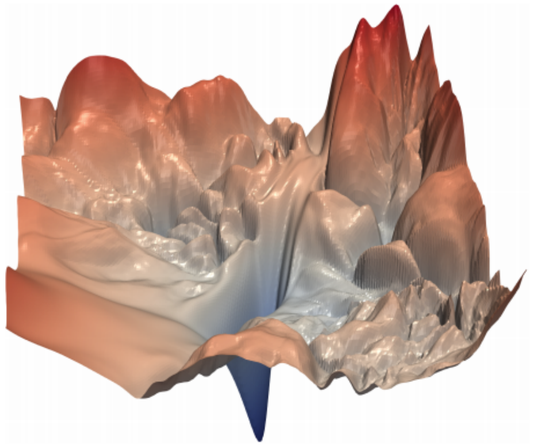
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

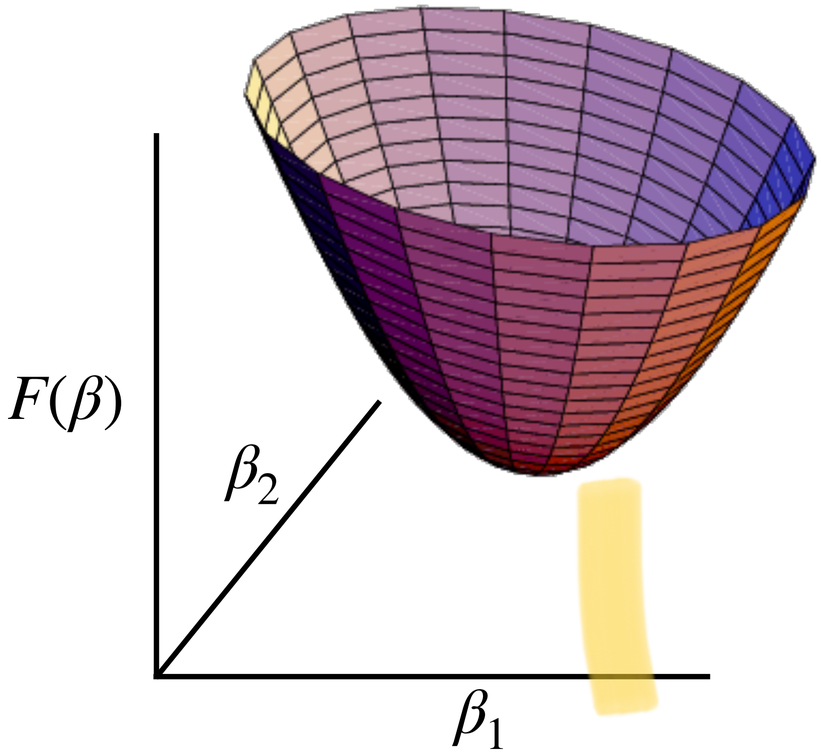
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) .$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



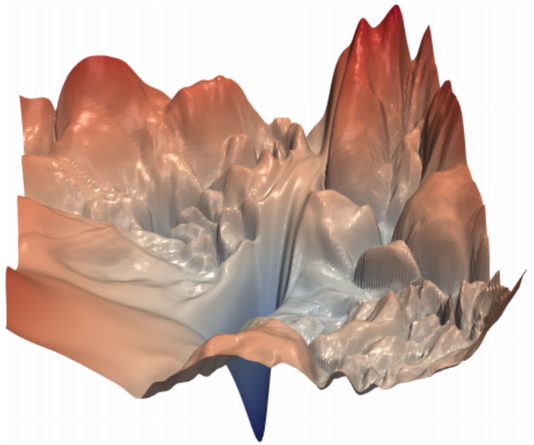
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

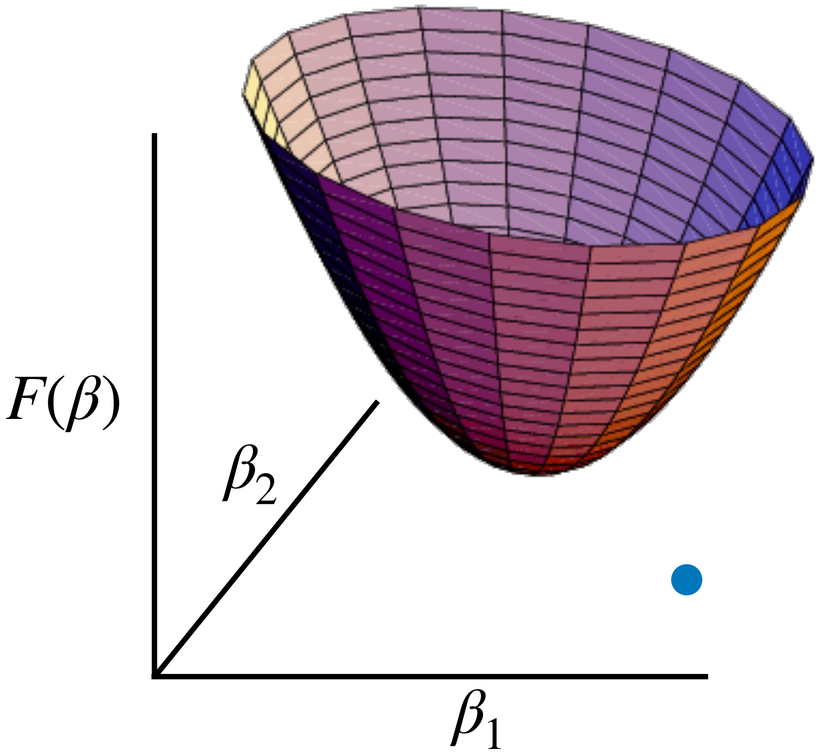
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



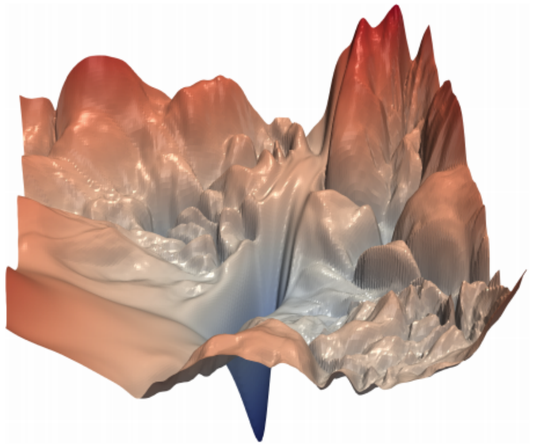
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

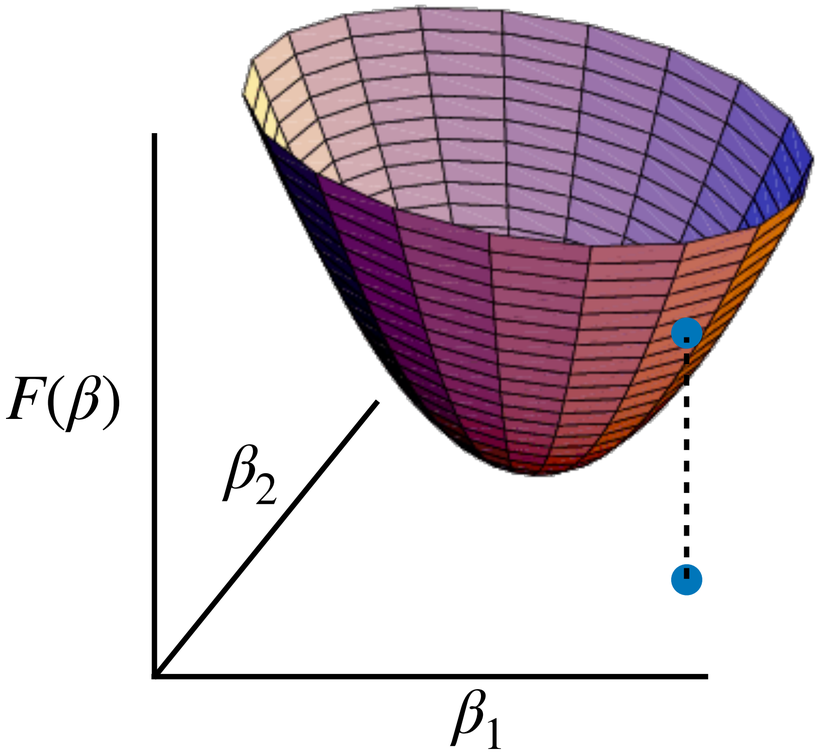
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



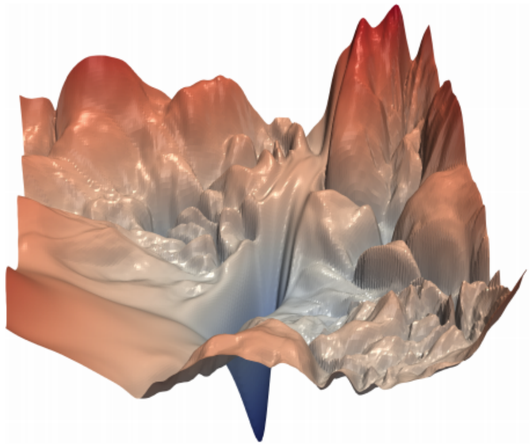
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

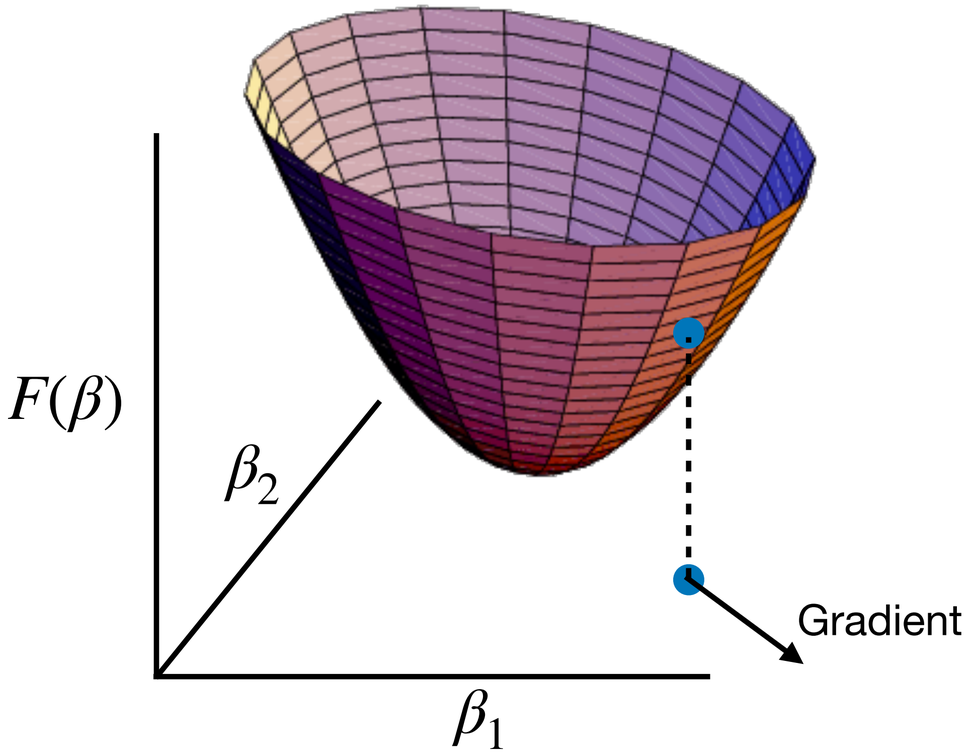
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



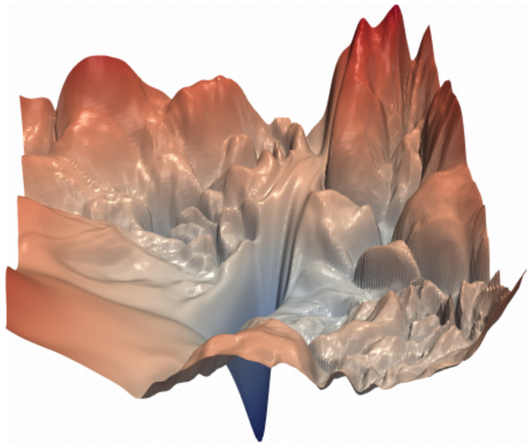
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

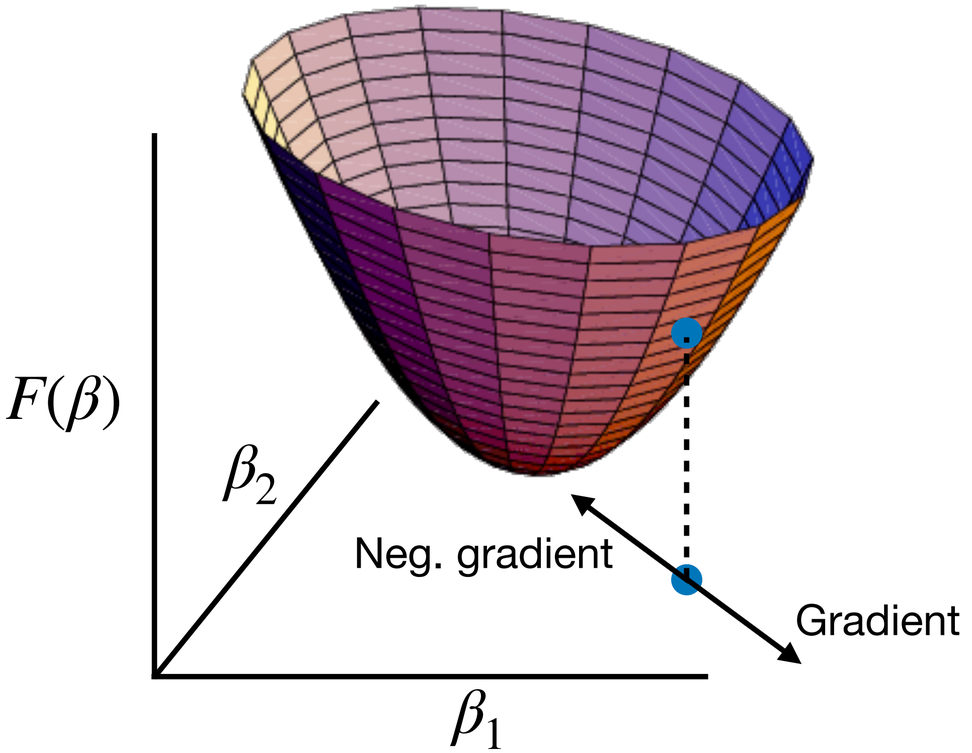
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



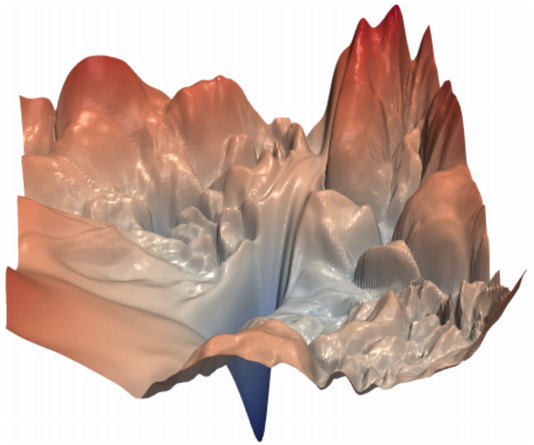
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

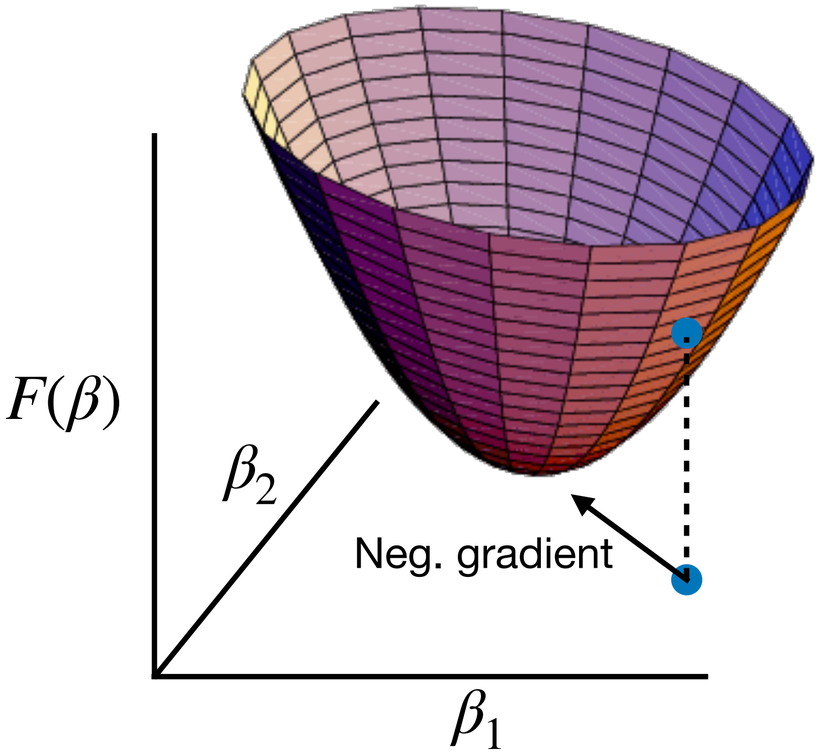
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



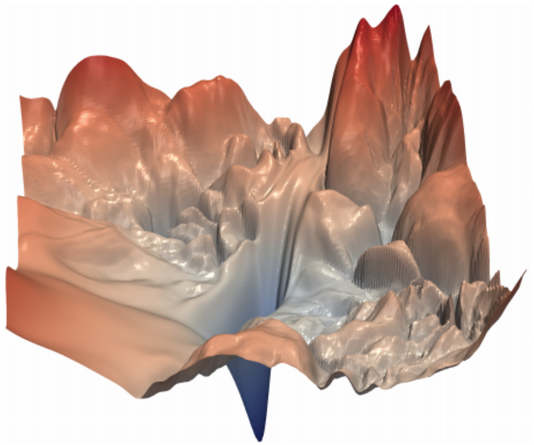
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

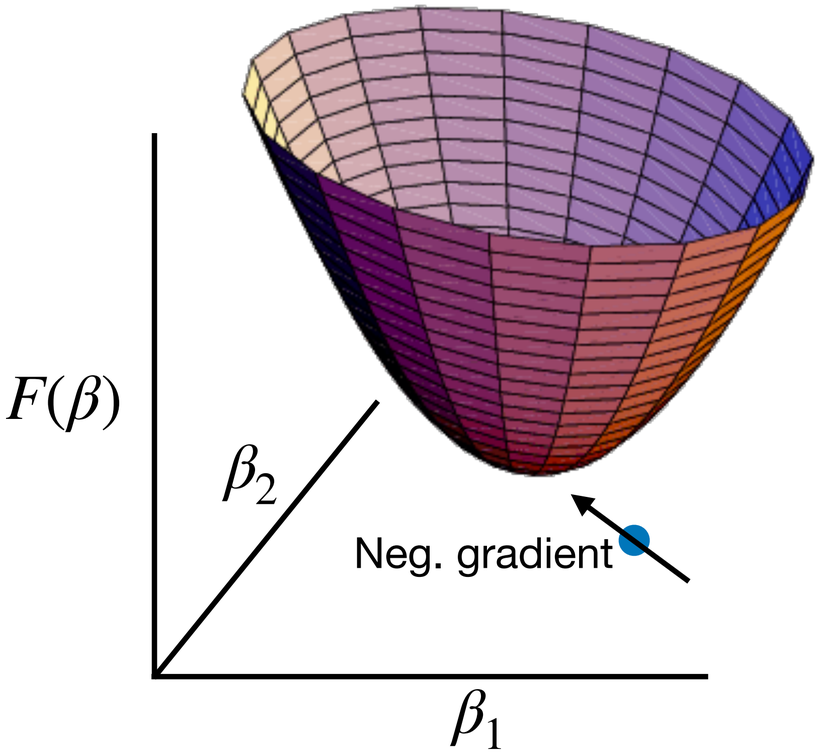
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



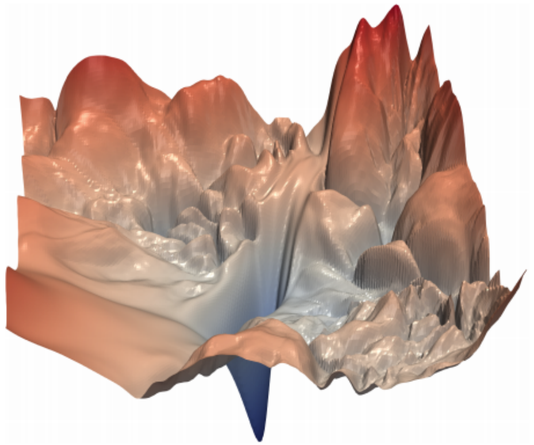
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

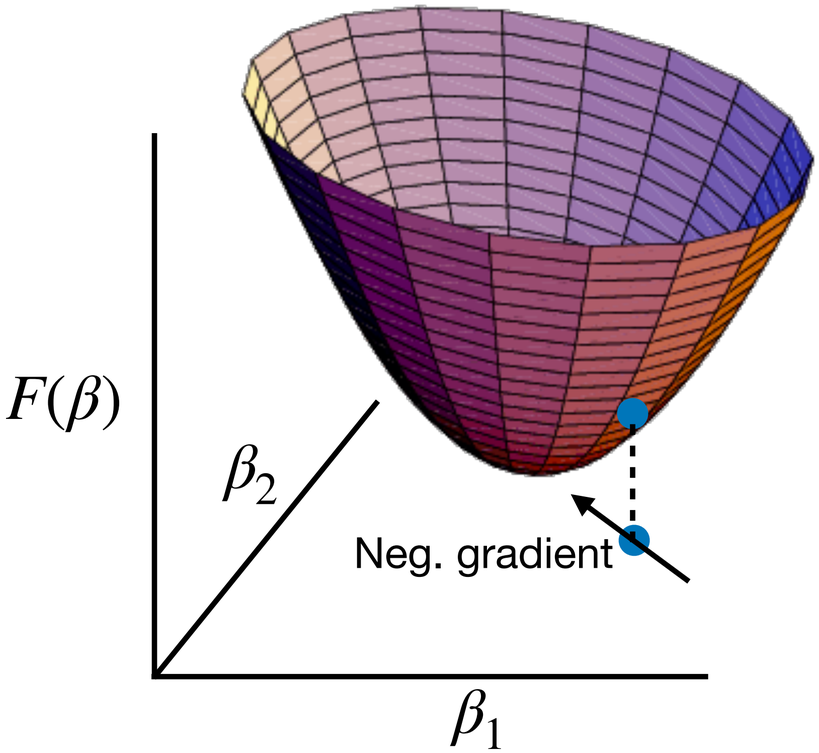
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



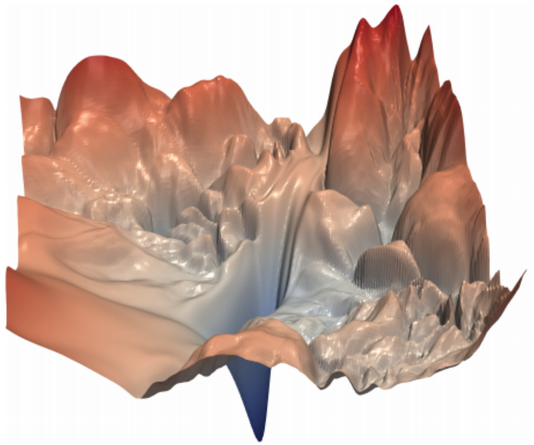
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

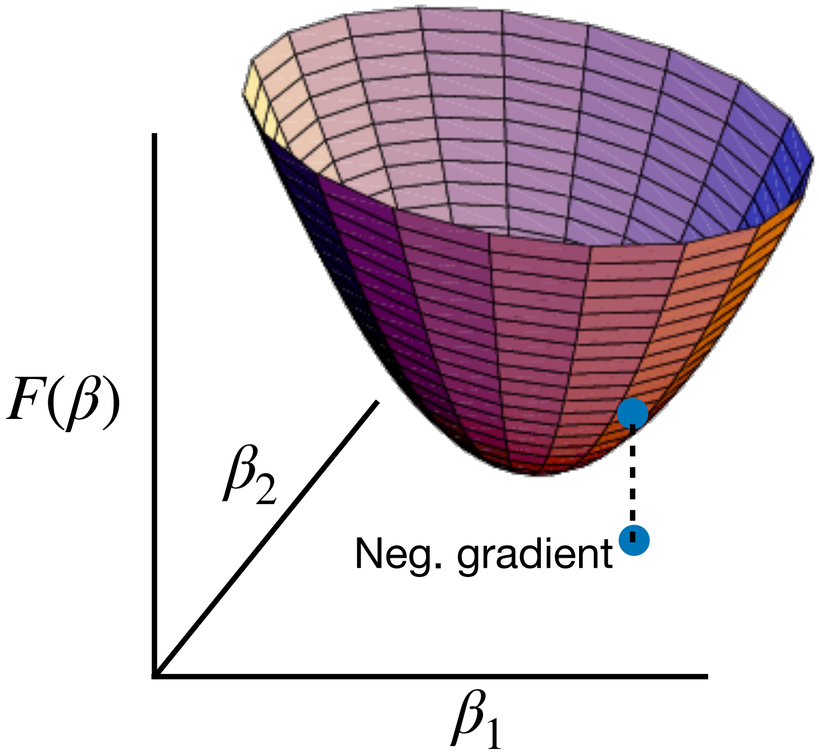
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



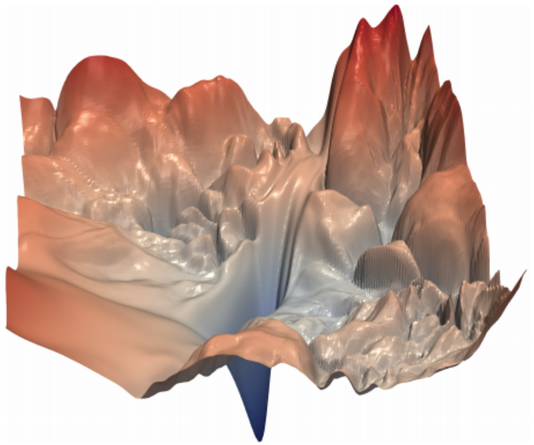
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

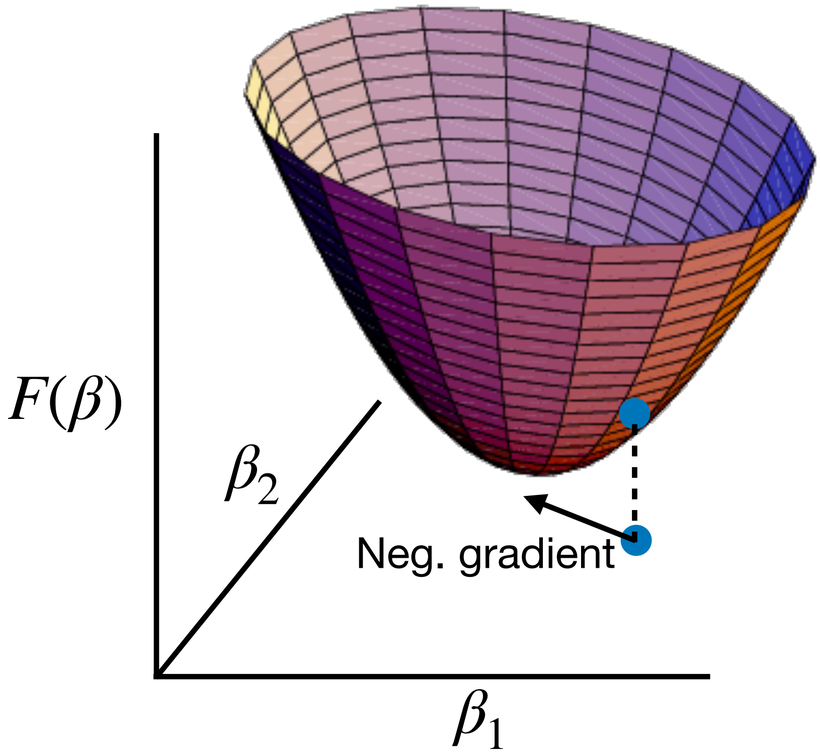
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



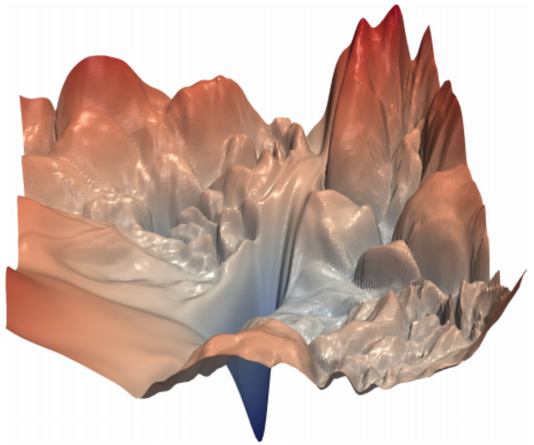
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

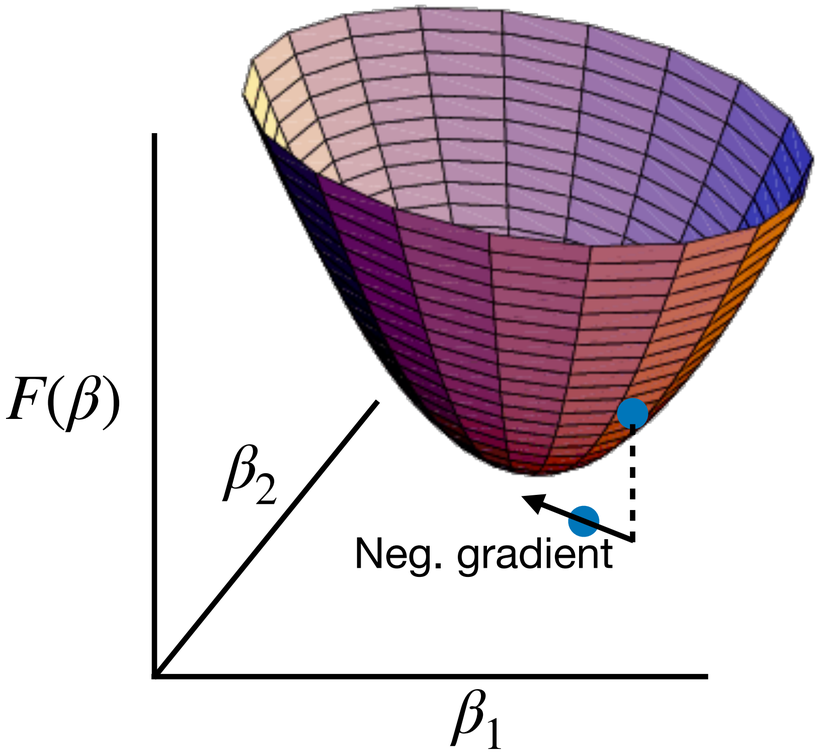
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



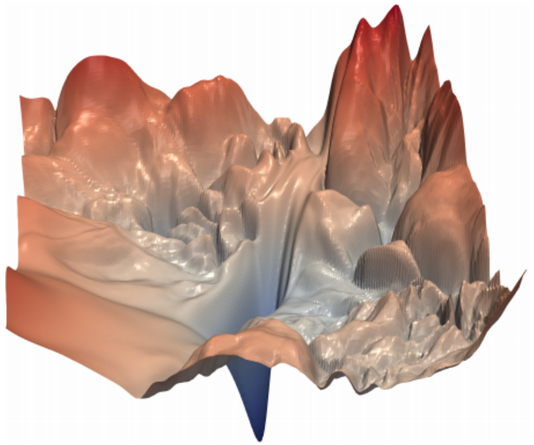
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

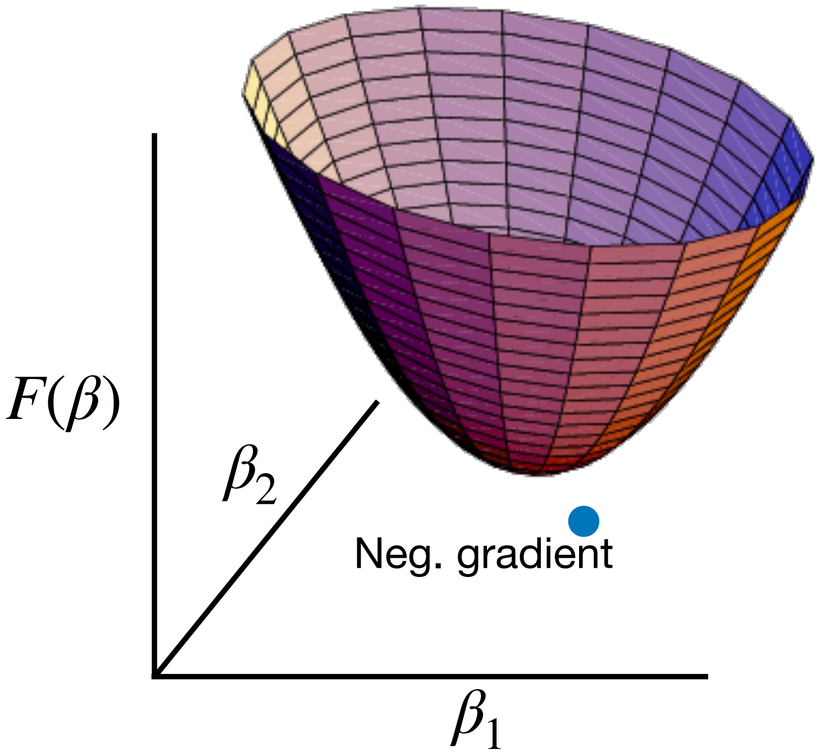
$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



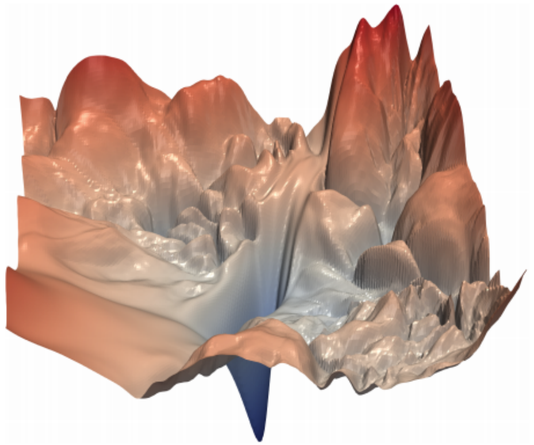
Training deep learning models

Training a neural network amounts to solving our usual optimization problem:

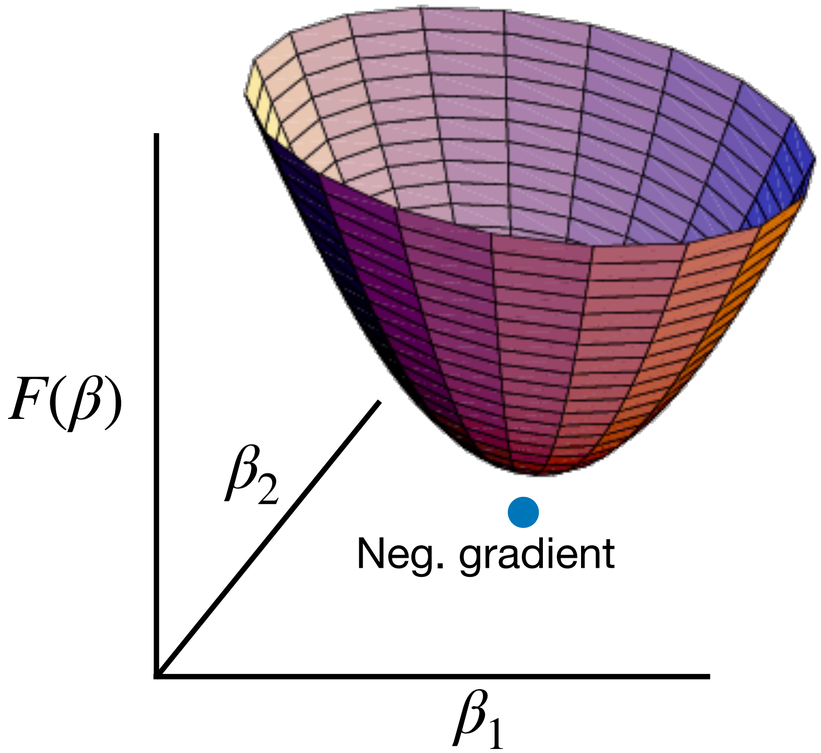
$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} L(Y_i, f_\beta(X_i)) + \lambda \cdot \text{penalty}(\beta) = \arg\min_{\beta} F(\beta) \,.$$

Unfortunately, neural networks have non-convex objective functions, causing issues discussed in the previous lecture.

In practice, deep learning models are trained using stochastic gradient descent, which is a variant of gradient descent.



https://arxiv.org/abs/1712.09913



The computational cost of gradient descent

unfortunately, super costly → computing once requires iterating over entire data set

computational cost = (cost of computing the gradient) × (number of iterations)

Number of iterations has to do with the size of the learning rate (like boosting) and the shape of the objective function F.

Cost of computing the gradient has to do with the size of the training data:

$$\text{If } F(\beta) = \frac{1}{n}\sum_{i=1}^{n} L(Y_i, f_\beta(X_i)), \text{ then } \nabla F(\beta) = \frac{1}{n}\sum_{i=1}^{n} \nabla L(Y_i, f_\beta(X_i)).$$

Each evaluation of the gradient requires a pass through the entire training data, and modern data sets can have millions of training observations.

This can make gradient descent prohibitively expensive.

Stochastic gradient descent

Use subset $S \subseteq \{1,\ldots,n\}$ (mini-batch) of observations to approximate gradient:

$$\nabla F(\beta) = \frac{1}{n}\sum_{i=1}^{n} \nabla L(Y_i, f_\beta(X_i)) \approx \frac{1}{|S|}\sum_{i\in S} \nabla L(Y_i, f_\beta(X_i)).$$

1. Choose some initial value of β.
2. Randomly assign observations to mini-batches $S_1, \ldots, S_M$ of a certain size
3. Repeat until convergence:
   - For $m = 1,\ldots,M$, update $\beta \leftarrow \beta - \gamma \cdot \frac{1}{|S_m|}\sum_{i\in S_m} \nabla L(Y_i, f_\beta(X_i))$.

One "epoch"

Stochastic gradient descent

Use subset $S \subseteq \{1,\ldots,n\}$ (mini-batch) of observations to approximate gradient:

$$\nabla F(\beta) = \frac{1}{n}\sum_{i=1}^{n} \nabla L(Y_i, f_\beta(X_i)) \approx \frac{1}{|S|}\sum_{i\in S} \nabla L(Y_i, f_\beta(X_i)).$$

1. Choose some initial value of β.
2. Randomly assign observations to mini-batches $S_1, \ldots, S_M$ of a certain size
3. Repeat until convergence:
   - For $m = 1,\ldots,M$, update $\beta \leftarrow \beta - \gamma \cdot \frac{1}{|S_m|}\sum_{i\in S_m} \nabla L(Y_i, f_\beta(X_i))$.

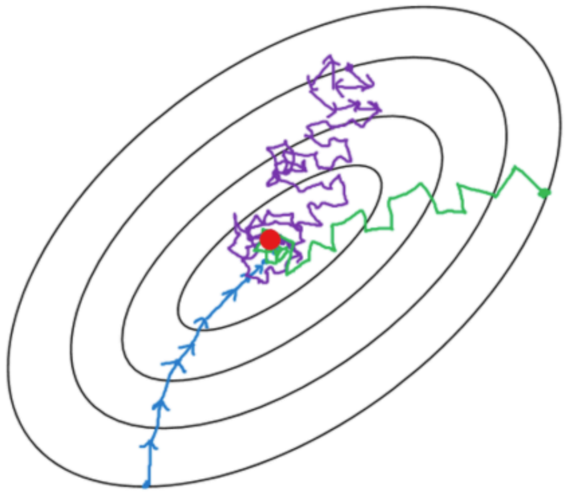
One "epoch"

compute each observation & then we need to sum up

Backpropagation: An efficient algorithm to compute $\nabla L(Y_i, f_\beta(X_i))$

Behavior of stochastic gradient descent

Stochastic gradient descent wobbles toward decreasing values of the objective:



Batch gradient descent (Mini-batch size = n)
Mini-batch gradient Descent (1 < mini-batch size < n)
Stochastic gradient descent (Mini-batch size = 1)

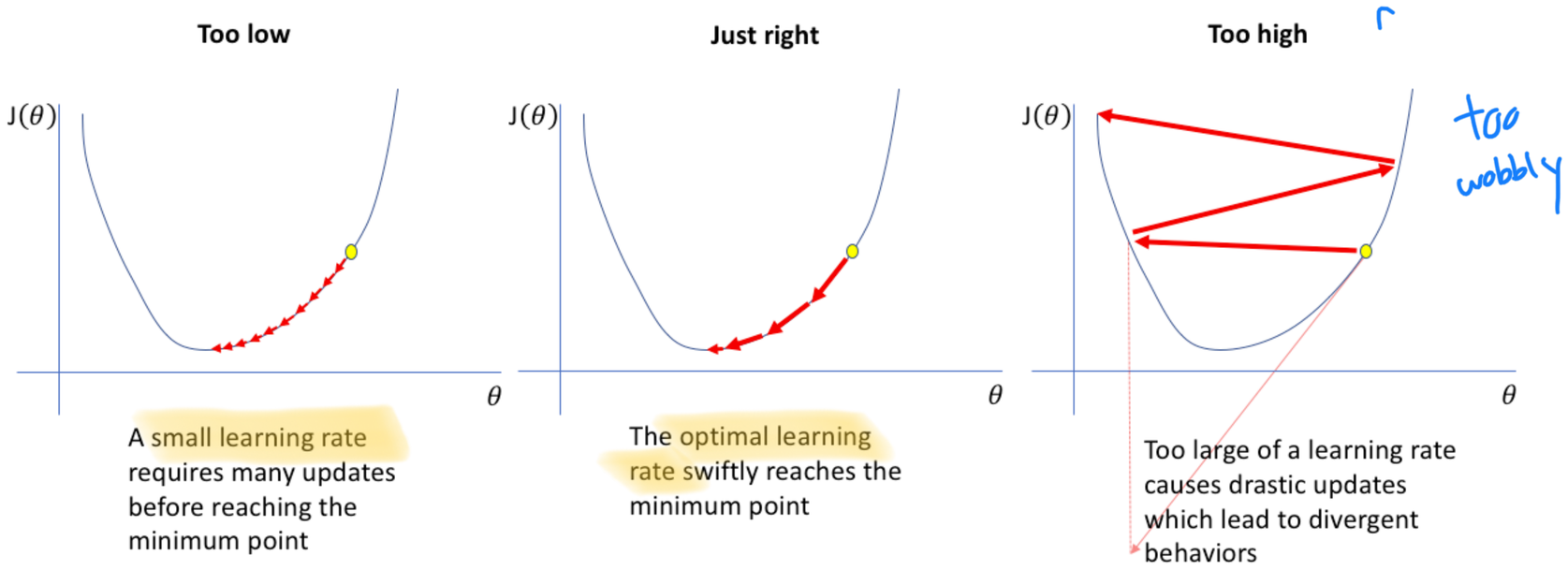
↳ wobbling a lot, but cost of steps is cheaper

Source: https://medium.com/analytics-vidhya/gradient-descent-vs-stochastic-gd-vs-mini-batch-sgd-fbd3a2cb4ba4

The smaller the mini-batch, the cheaper and more wobbly each step is;
Intermediate mini-batch sizes tend to work well, e.g. mini-batch size = 32.

Bonus: The extra randomness sometimes allows SGD to wobble past local minima.

The learning rate for (stochastic) gradient descent



Source: https://www.jeremyjordan.me/nn-learning-rate/

- Setting the learning rate is more of an art than a science; might need to try a few values to get a good one.
- Especially for non-convex optimization, people come up with clever strategies like shrinking learning rates, cycling learning rates, adaptive learning rates, etc. (RMSprop, Adam, AdaGrad, AdaDelta, …)

→ ∅ worry too much about how they work

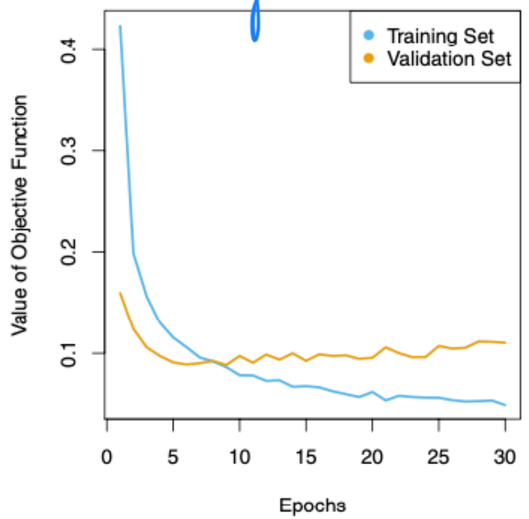
neural nets can still overfit

Regularization in deep learning models

Explicit regularization via penalization. Ridge regression penalty is the most common; this kind of penalization is known as weight decay.

↳ pretty much same as ridge

Implicit regularization: Other techniques to control complexity of the model.

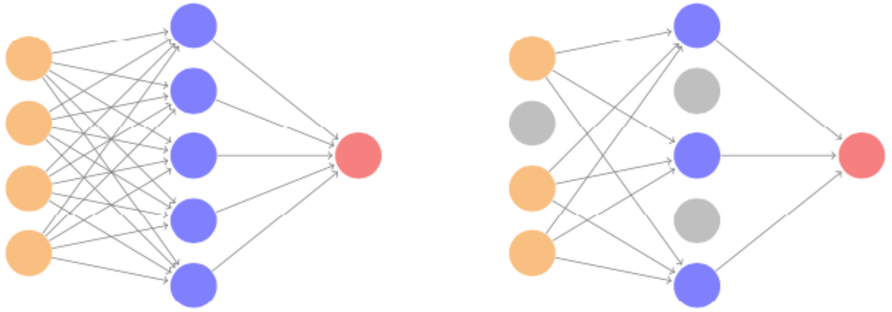


- Early stopping: Do not run SGD until convergence; rather, stop SGD when validation error starts to increase.

do not go all the way, just part!

- Dropout: At each SGD iteration, remove a randomly selected set of nodes from the network (analogous to sub-sampling features for random forests).

randomly selects nodes to be dropped



Model complexity and model tuning in deep learning

Deep learning has many tuning parameters:

- Network architecture (number of layers, units per layer)
- Optimization algorithm used (RMSprop, Adam, etc.)
- Weight decay parameter, dropout rate, number of SGD iterations

mostly in terms of classification, but you can also use for regression

↳ what % dropouts

More complex models arise from bigger networks, less weight decay, less dropout, more SGD iterations.

Deep learning performance can be sensitive to these parameters; there is no standard way to tune neural networks; tuning is computationally expensive.

For computational savings, validation sets are typically employed instead of cross-validation to tune neural networks.