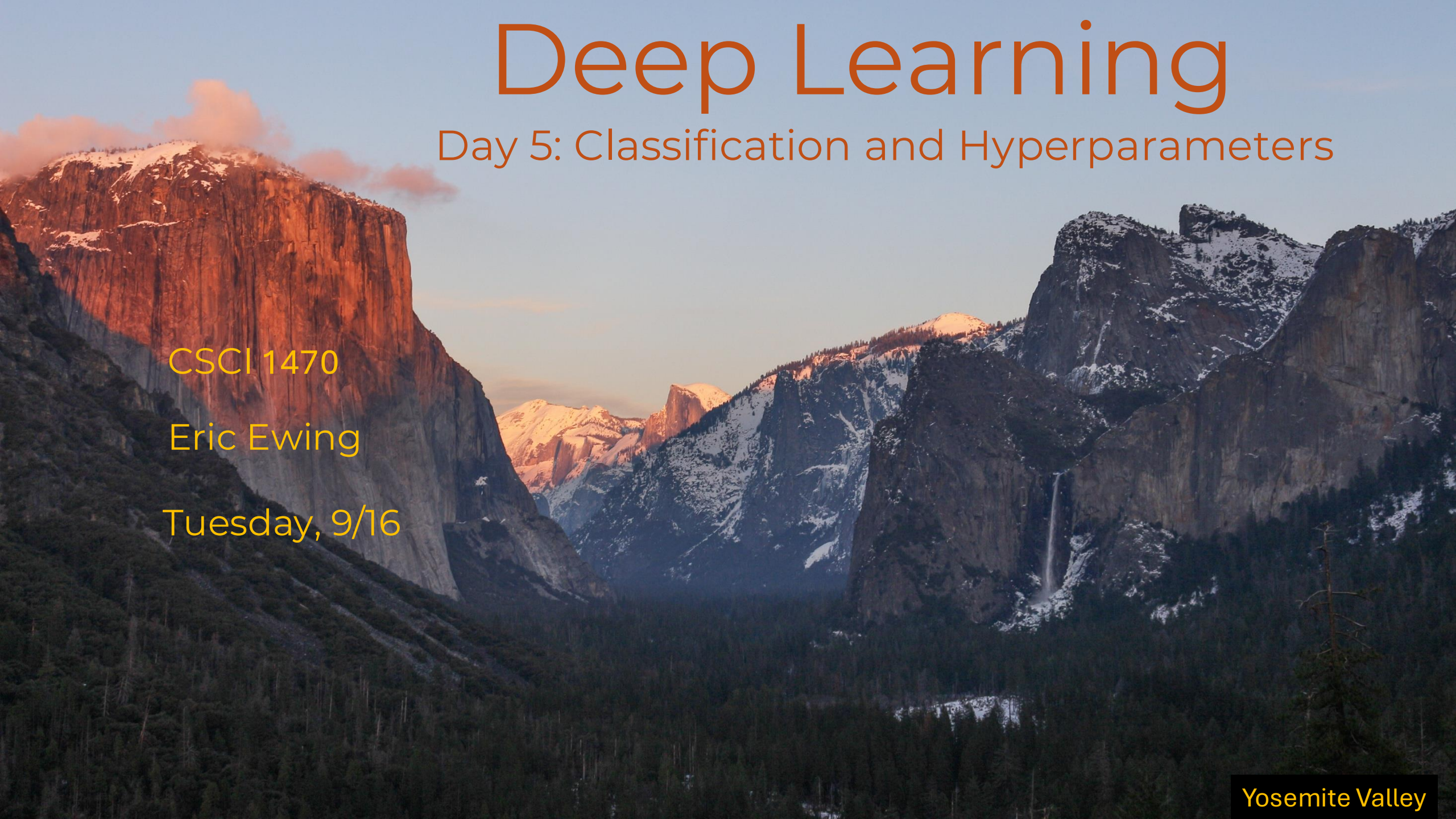
Deep Learning

Day 5: Classification and Hyperparameters

CSCI 1470

Eric Ewing

Tuesday, 9/16



Yosemite Valley

HW 3: Beras

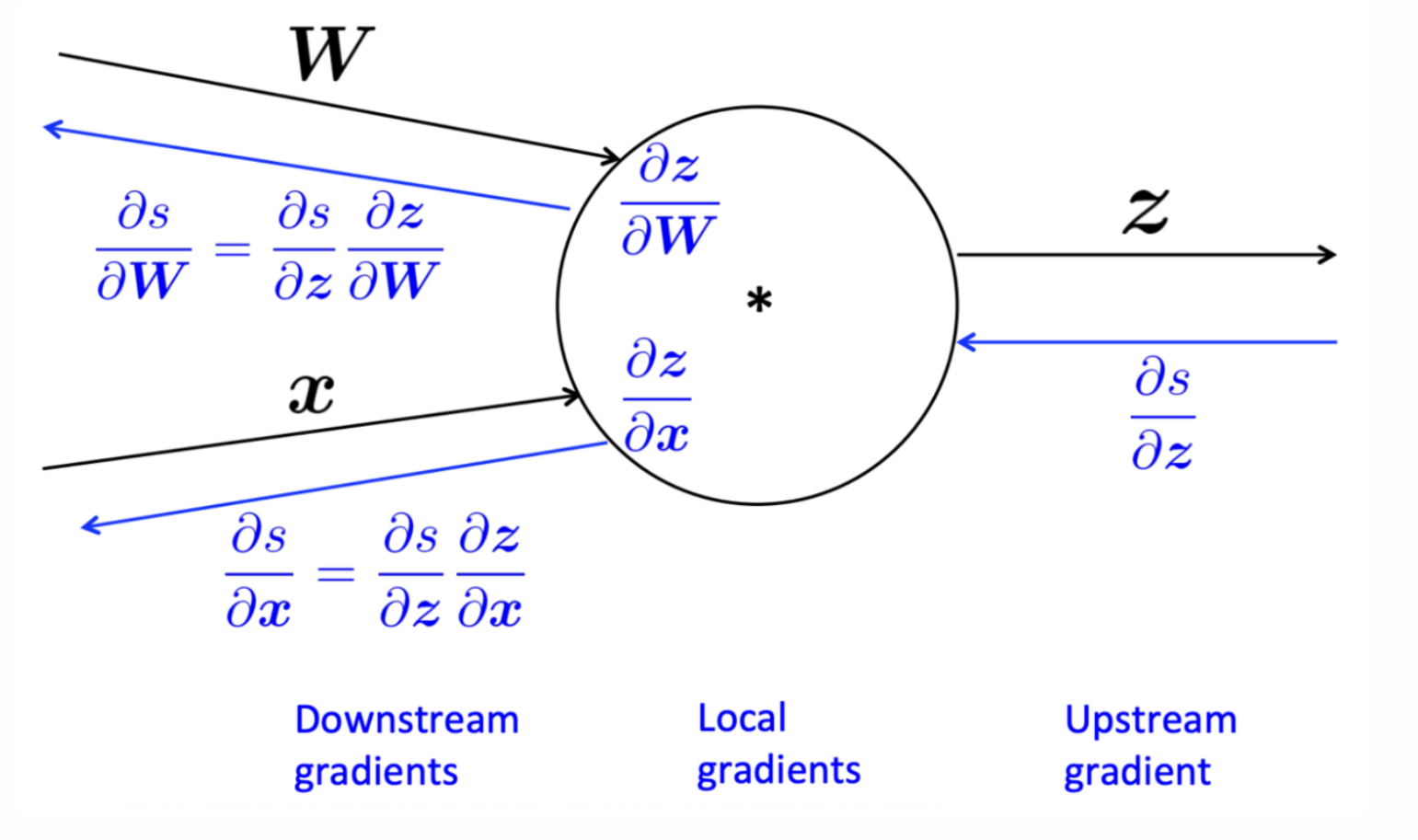
Will come out today! (conceptual component due in 1 week, programming in 2 weeks)

Implement neural network with numpy (i.e., weight layers, activations, loss functions) and training (i.e., gradient tape for backprop and optimizers)

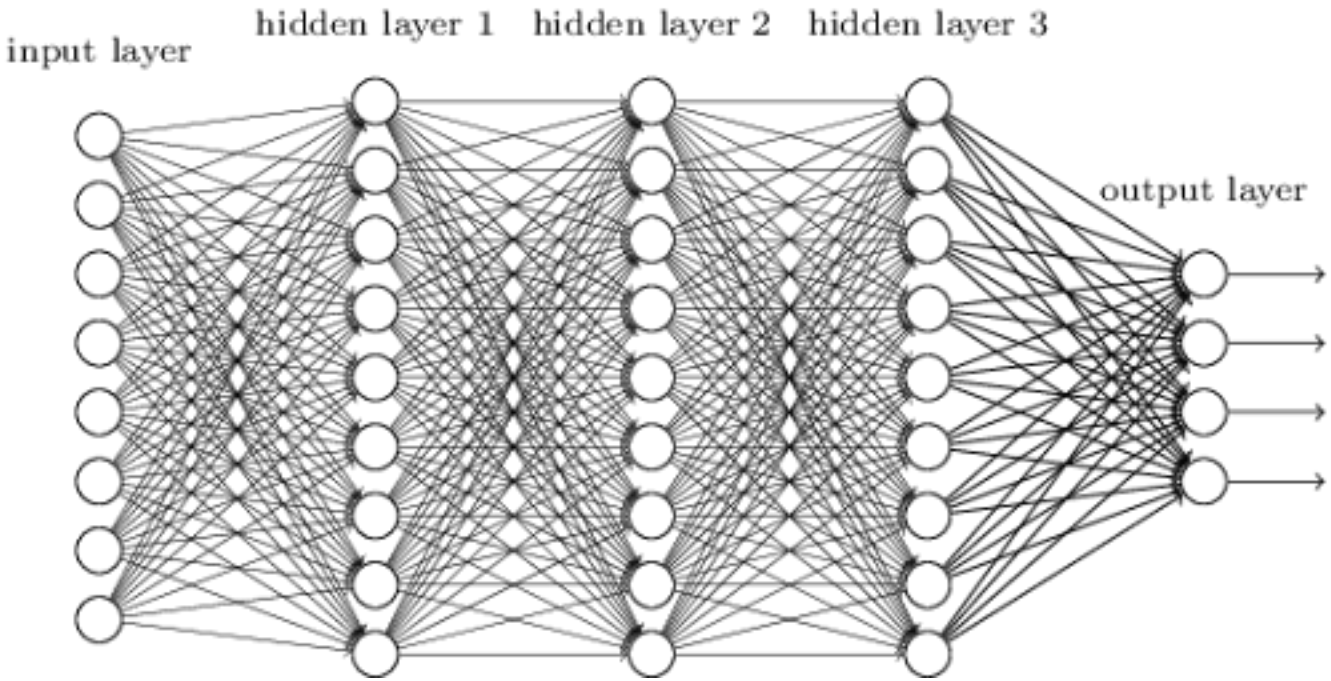
Unlimited submissions during the first week, limited to 15 submissions in week 2. (Additional submissions will still be graded, but test results won’t be visible)

Beras Companion Guide

Linked in handout, set of companion notes that contain additional explanation of functions and classes.

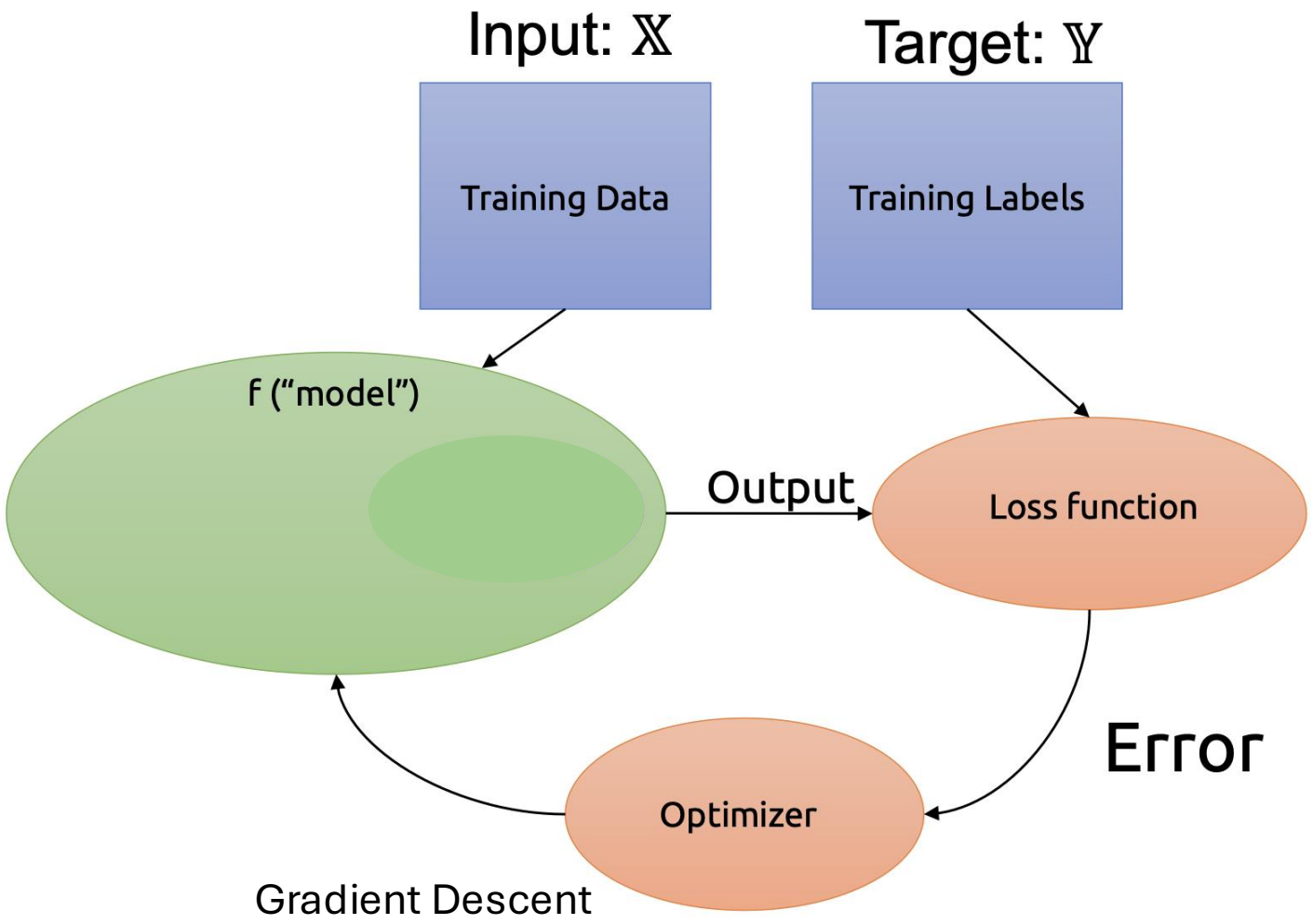


Recap: Neural Networks (MLPs)



Each neuron is the weighted sum of inputs, a bias, and an activation function

“Classic” Supervised Learning in Machine Learning



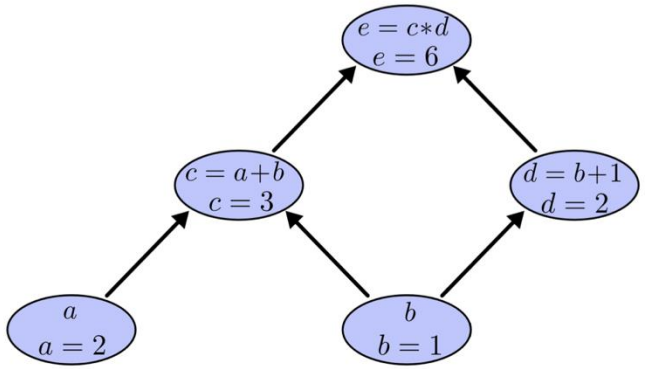
Recap

Train models with gradient descent

Find gradient using backpropagation and compute graphs

Computation Graph

$$e = (a + b) \cdot (b + 1)$$



Why should you care about compute graphs?

(This is much more of a common issue in pytorch than tensorflow)

```
def train_with_memory_leak():
    running_loss = 0.0
    for epoch in range(100):
        for i, (inputs, targets) in enumerate(loader):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss

            if i % 10 == 9:
                print(f'Loss: {running_loss / 10}')
                running_loss = 0.0
```

Why should you care about compute graphs?

(This is much more of a common issue in pytorch than tensorflow)

Running loss' compute graph will contain the compute graph of loss!

```python
def train_with_memory_leak():
    running_loss = 0.0
    for epoch in range(100):
        for i, (inputs, targets) in enumerate(loader):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss

            if i % 10 == 9:
                print(f'Loss: {running_loss / 10}')
                running_loss = 0.0
```

Why should you care about compute graphs?

(This is much more of a common issue in pytorch than tensorflow)

```python
def train_with_memory_leak():
    running_loss = 0.0
    for epoch in range(100):
        for i, (inputs, targets) in enumerate(loader):
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss

            if i % 10 == 9:
                print(f'Loss: {running_loss / 10}')
                running_loss = 0.0
```

Running loss' compute graph will contain the compute graph of loss!

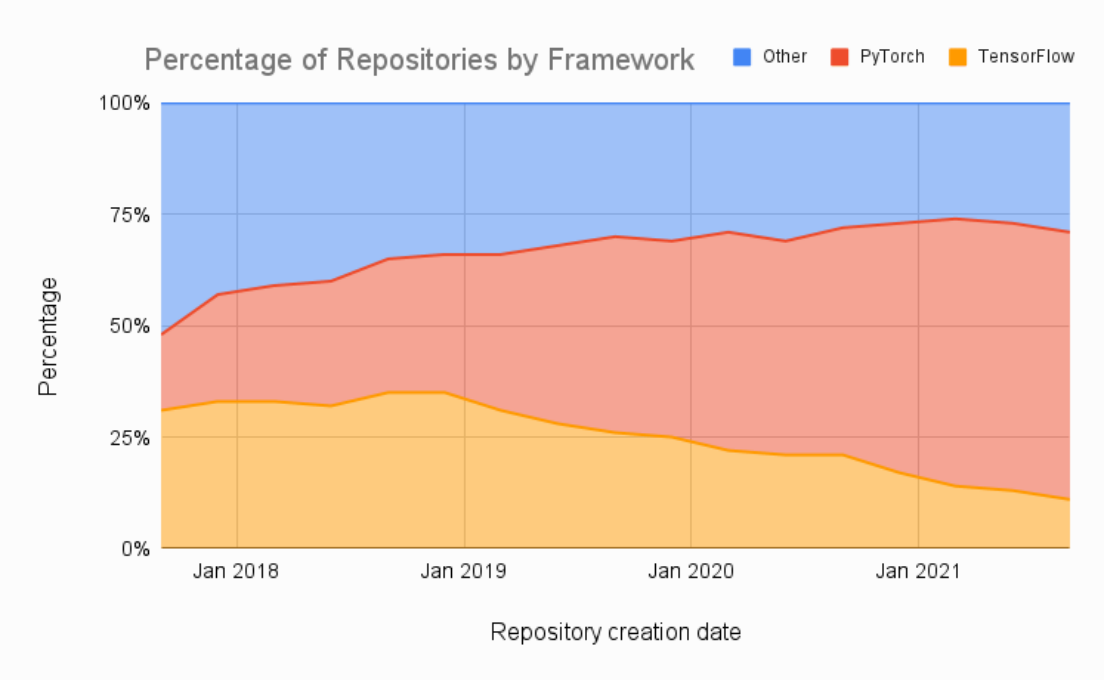
The memory required to store running_loss will only ever increase!

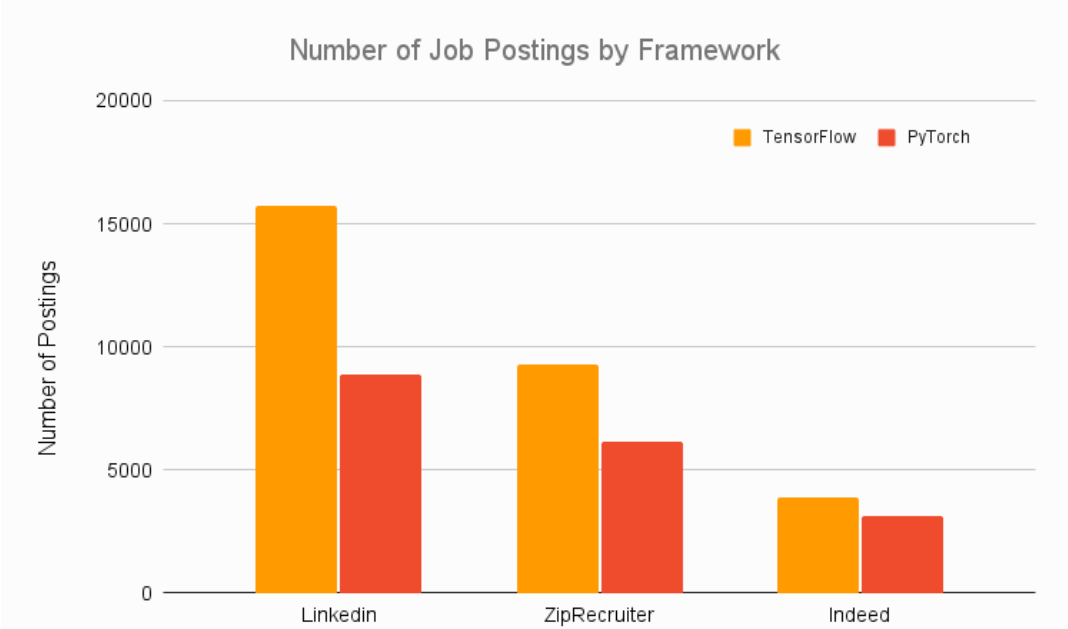
DL Frameworks

PyTorch

TensorFlow

- Main current frameworks are Tensorflow, Pytorch, and Jax
- TF and torch are becoming increasingly similar in style and performance
- Jax is new and different





https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/

Tensorflow

Tensorflow

- Developed and maintained by Google

Tensorflow

- Developed and maintained by Google
- In addition to autodiff features it also provides:
  - Many common functions (i.e., Softmax, Sigmoid, Cross Entropy, etc.)
  - An easy way to train models (**Keras**)
  - Strong support for hardware acceleration (i.e., if you have a GPU, TF will figure out how to use it)

Tensorflow

- Developed and maintained by Google
- In addition to autodiff features it also provides:
  - Many common functions (i.e., Softmax, Sigmoid, Cross Entropy, etc.)
  - An easy way to train models (Keras)
  - Strong support for hardware acceleration (i.e., if you have a GPU, TF will figure out how to use it)
- “Easier to deploy to production” (has been the general consensus previously, but other frameworks have caught up)

Tensorflow

- Developed and maintained by Google
- In addition to autodiff features it also provides:
  - Many common functions (i.e., Softmax, Sigmoid, Cross Entropy, etc.)
  - An easy way to train models (Keras)
  - Strong support for hardware acceleration (i.e., if you have a GPU, TF will figure out how to use it)
- “Easier to deploy to production” (has been the general consensus previously, but other frameworks have caught up)
- TF lite for on device applications (e.g., phones)

Pytorch

Pytorch

- Developed by Facebook AI (now Meta)

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes
- No Gradient Tape, each tensor (matrix/vector) is “trainable” or not. If a tensor is trainable then all operations on it are tracked.

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes
- No Gradient Tape, each tensor (matrix/vector) is “trainable” or not. If a tensor is trainable then all operations on it are tracked.
- Slightly more work to use GPUs or other hardware

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes
- No Gradient Tape, each tensor (matrix/vector) is “trainable” or not. If a tensor is trainable then all operations on it are tracked.
- Slightly more work to use GPUs or other hardware
- Harder to track stats
  - (I still use TF’s tensorboard stat tracker when using Pytorch)

Pytorch

- Developed by Facebook AI (now Meta)
- More common in the research and academic community
- “More flexible” and easier to write custom backward passes
- No Gradient Tape, each tensor (matrix/vector) is “trainable” or not. If a tensor is trainable then all operations on it are tracked.
- Slightly more work to use GPUs or other hardware
- Harder to track stats
  - (I still use TF’s tensorboard stat tracker when using Pytorch)
- Easier to learn and use than tensorflow
  - Better error reporting, training code is harder to write but easier to debug

Jax

Jax

- Also developed by Google…

Jax

- Also developed by Google…
- Very new compared to Pytorch and Tensorflow

Jax

- Also developed by Google…
- Very new compared to Pytorch and Tensorflow
- ***Much Faster***

Jax

- Also developed by Google…
- Very new compared to Pytorch and Tensorflow
- ***Much Faster***
- Takes advantage of Just In Time (JIT) compiling to speed up execution

Jax

- Also developed by Google…
- Very new compared to Pytorch and Tensorflow
- ***Much Faster***
- Takes advantage of Just In Time (JIT) compiling to speed up execution
- Functional programming paradigm

Improving Gradient Descent

Improving Gradient Descent

Computing the full gradient for a large dataset takes a very long time and it often will not fit in memory, slowing it down even further

Improving Gradient Descent

Computing the full gradient for a large dataset takes a very long time and it often will not fit in memory, slowing it down even further

Solution: Approximate the gradient by sampling a selection of examples (i.e., a batch). Run a gradient descent step with that batch

Stochastic Gradient Descent

For N epochs:

- sample a batch B from dataset X
- compute predictions and loss function
- compute gradient
- update weights with small step in direction of negative grad.

Stochastic Gradient Descent

For N epochs:

- sample a batch B from dataset X
- compute predictions and loss function
- compute gradient
- update weights with small step in direction of negative grad.

Training is non-deterministic because batches are sampled randomly from dataset

Stochastic Gradient Descent

For N epochs:

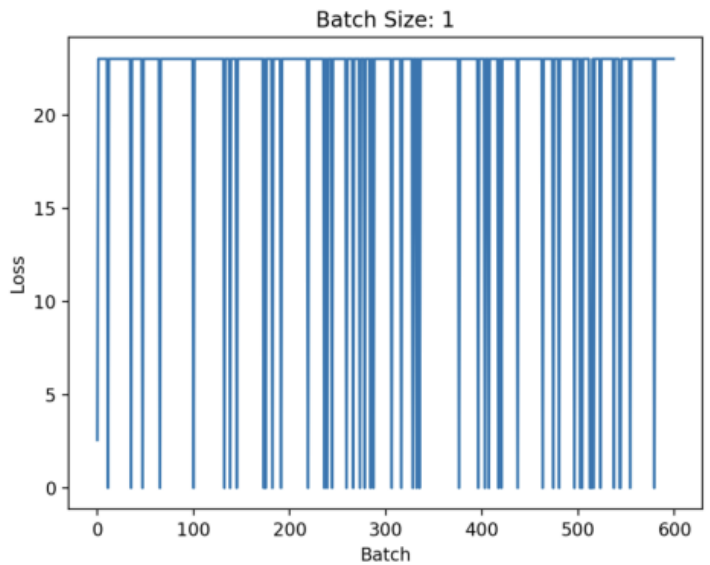
- sample a batch B from dataset X
- compute predictions and loss function
- compute gradient
- update weights with small step in direction of negative grad.

Training is non-deterministic because batches are sampled randomly from dataset

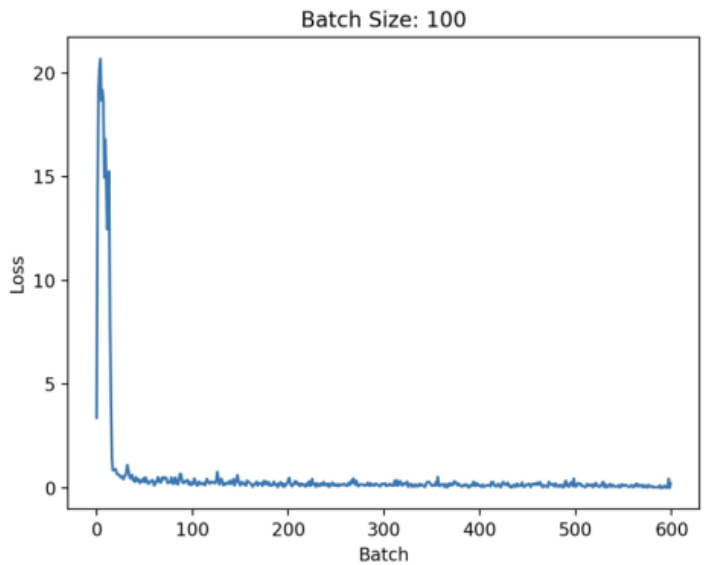
Why does this work? The expectation of the gradient is equal to the gradient itself!

What size should the batch be?

Small batch size:
Fast, jittery updates



Large batch size:
Slower, stable updates



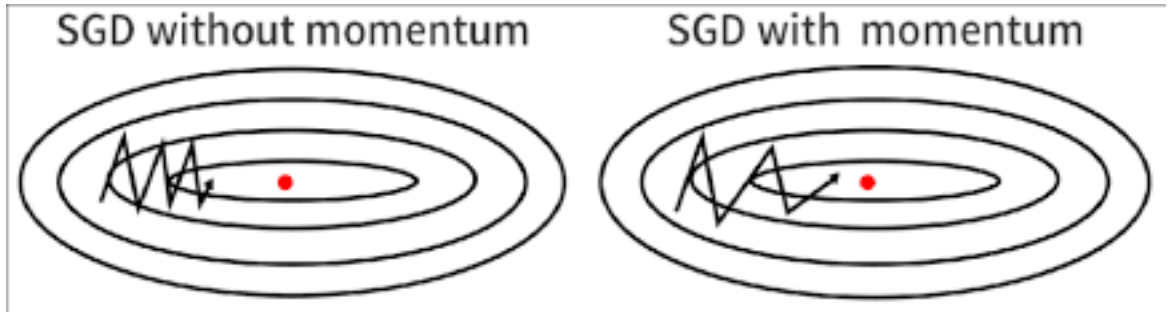
- Empirically, modern optimizers can handle larger batch size well
- Try to pick the largest batch size you can fit on your GPU!

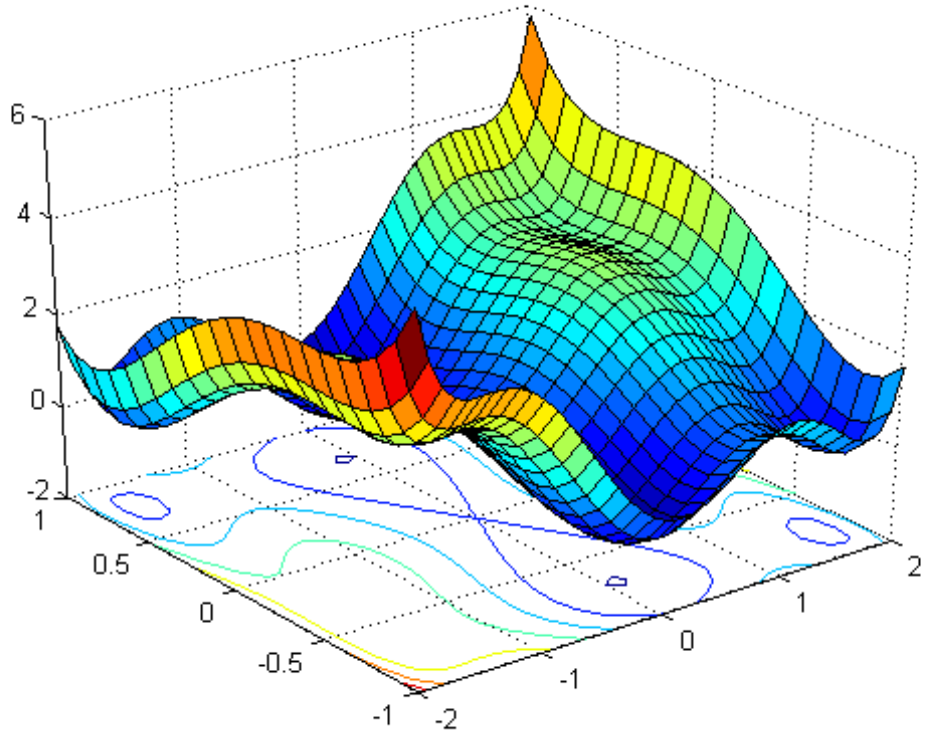
Further Improvements

$$\Delta w := \alpha \Delta w - \eta \nabla Q_i(w)$$
$$w := w + \Delta w$$

If gradient descent is like a ball rolling down a hill… What is that ball's mass?

SGD can be further improved by adding momentum term

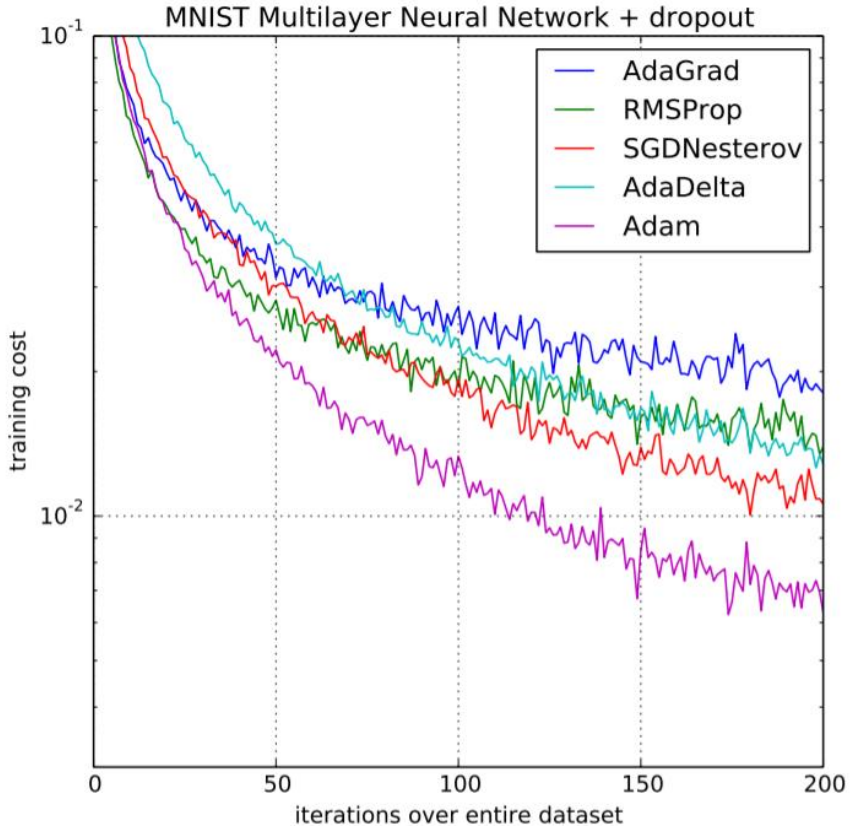




AdaM: SGD + Adaptive Momentum

Generally recommended as the best performing and easiest to use optimizer!

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)



Classification

Classification

In general, we’d like to optimize the accuracy of our model (#correct/#total)

Classification

In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Classification

In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Proposed Loss Function: $L = 1 - \frac{\#\ Correct}{n}$

Classification

In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Proposed Loss Function: $L = 1 - \frac{\#\ Correct}{n}$

The Issue: most of the time, the gradient of this loss function is $\nabla \mathrm{L}_\theta = 0$

Classification

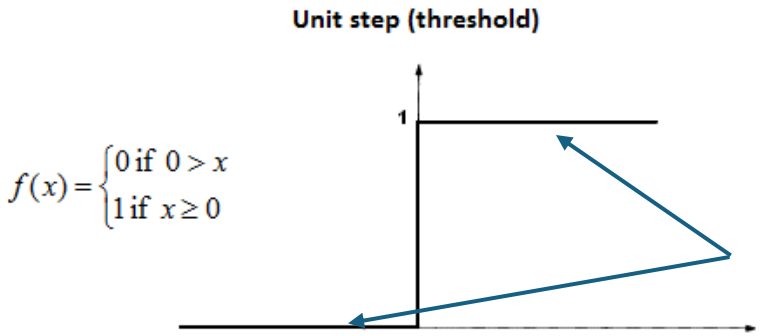
In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Proposed Loss Function: $L = 1 - \frac{\#\ Correct}{n}$

The Issue: most of the time, the gradient of this loss function is $\nabla \mathrm{L}_\theta = 0$

Gradient is only non-zero when changing a θ has an impact on output predictions



0 gradient everywhere except x=0

X=0 is not differentiable, but it does have a sub-gradient

Classification

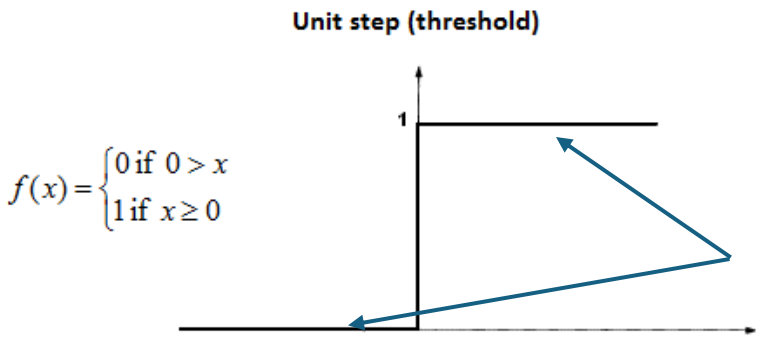
In general, we'd like to optimize the accuracy of our model (#correct/#total)

Need Loss function to be small for best model, not large.

Proposed Loss Function: $L = 1 - \frac{\#\ Correct}{n}$

The Issue: most of the time, the gradient of this loss function is $\nabla \mathrm{L}_\theta = 0$

Gradient is only non-zero when changing a θ has an impact on output predictions



We cannot use classification as a loss function because it is incompatible with gradient descent**. Understanding Gradients is key to understanding all decisions related to neural networks!**

0 gradient everywhere except x=0

X=0 is not differentiable, but it does have a sub-gradient

What is a reasonable loss function to use?

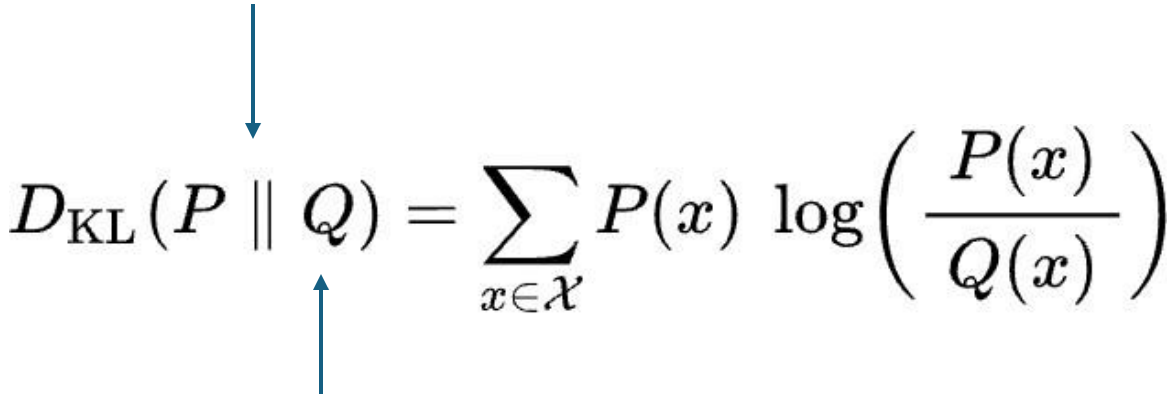
- Accuracy is a “hard” function
  - Hard to take meaningful derivatives of
- Other examples:
  - Max vs. Softmax
  - Ranking vs Softrank
  - Sign function (i.e., perceptron activation) vs. Softsign
  - Argmax

What is a reasonable loss function to use?

- Accuracy is a “hard” function
  - Hard to take meaningful derivatives of
- Other examples:
  - Max vs. Softmax
  - Ranking vs Softrank
  - Sign function (i.e., perceptron activation) vs. Softsign
  - Argmax

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions



$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

Defined for two probability distributions, P and Q

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

Defined for two probability distributions, P and Q

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Think of Q as what we predict and P as the ground truth Probabilities

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

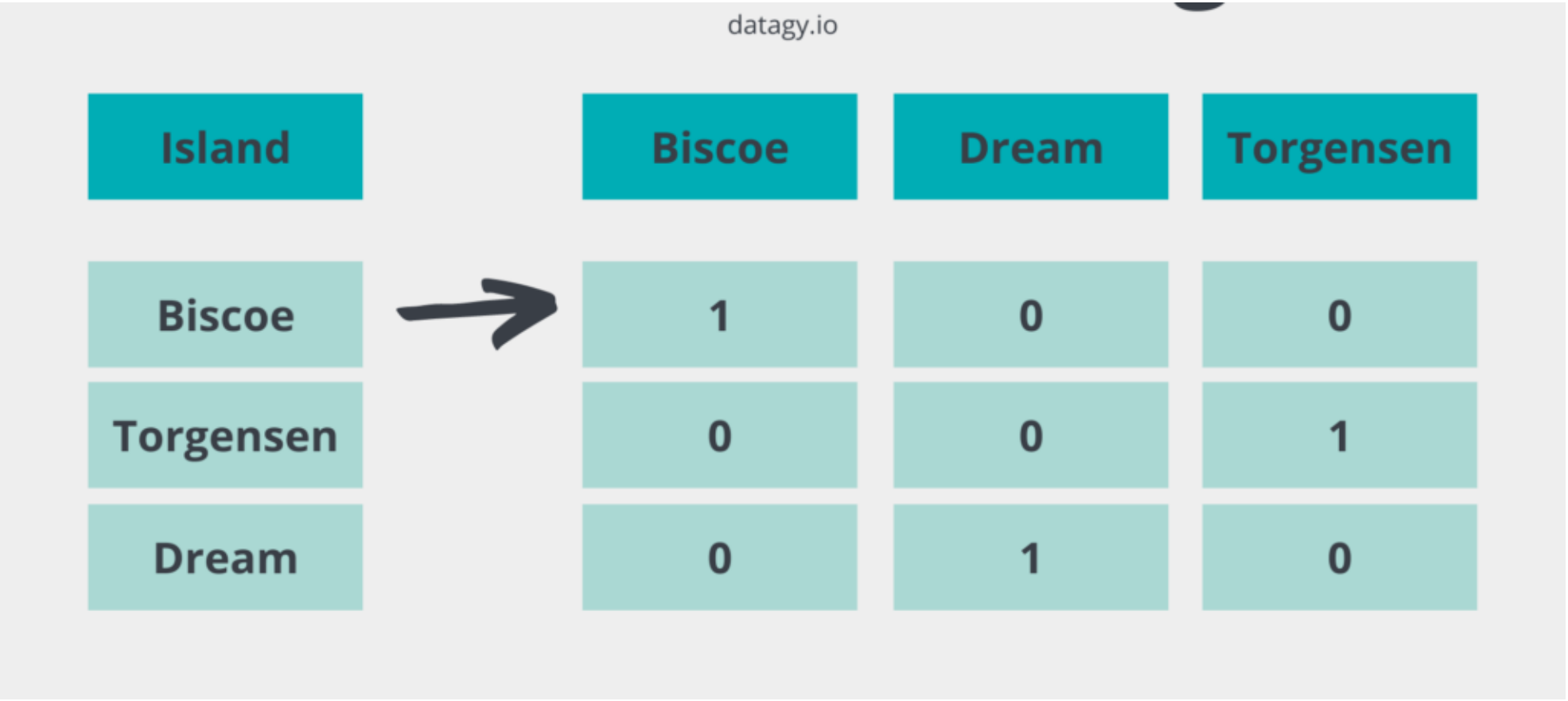
Defined for two probability distributions, P and Q

When P(x) is high, Q(x) should also be high… (Log(1) = 0)

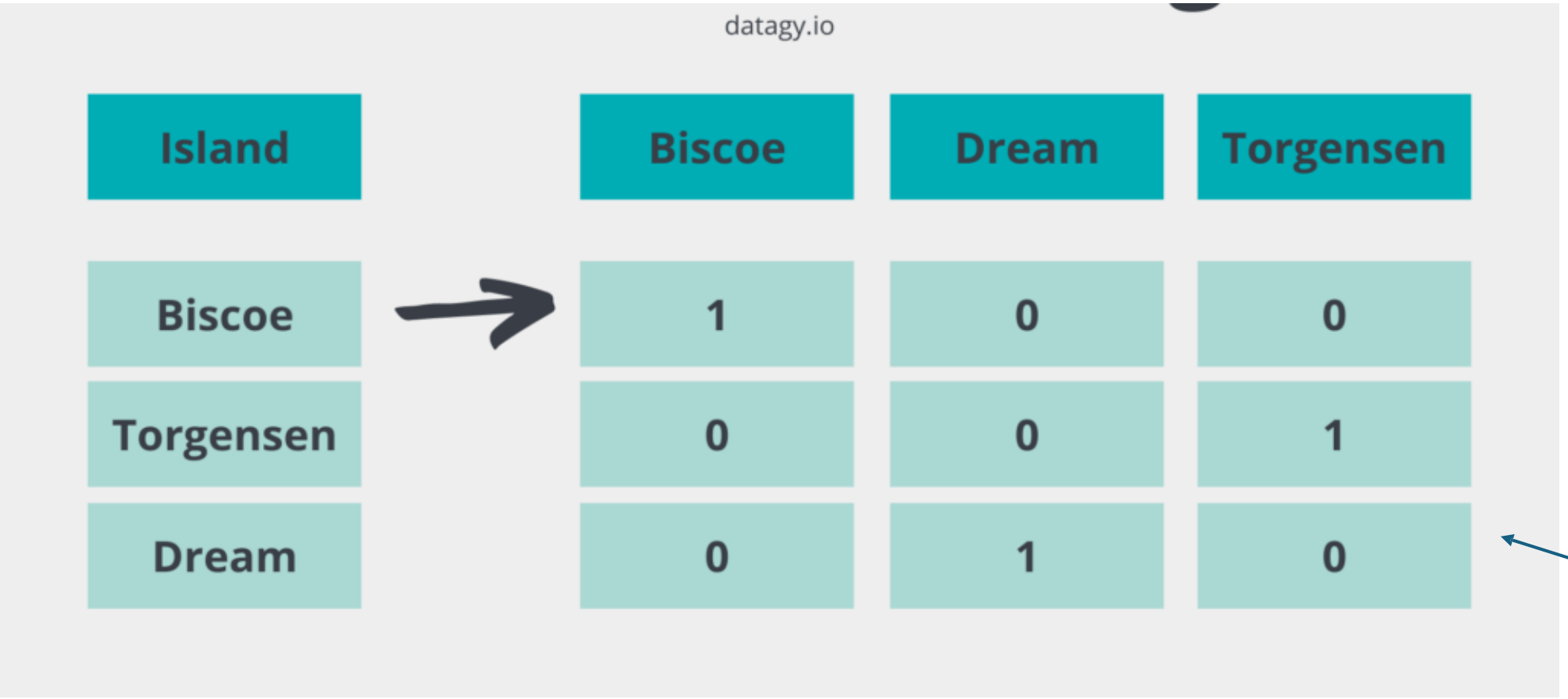
$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Think of Q as what we predict and P as the ground truth Probabilities

One-Hot Vectors Revisited



One-Hot Vectors Revisited



Can be interpreted as a probability!

Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Kullback–Leibler divergence

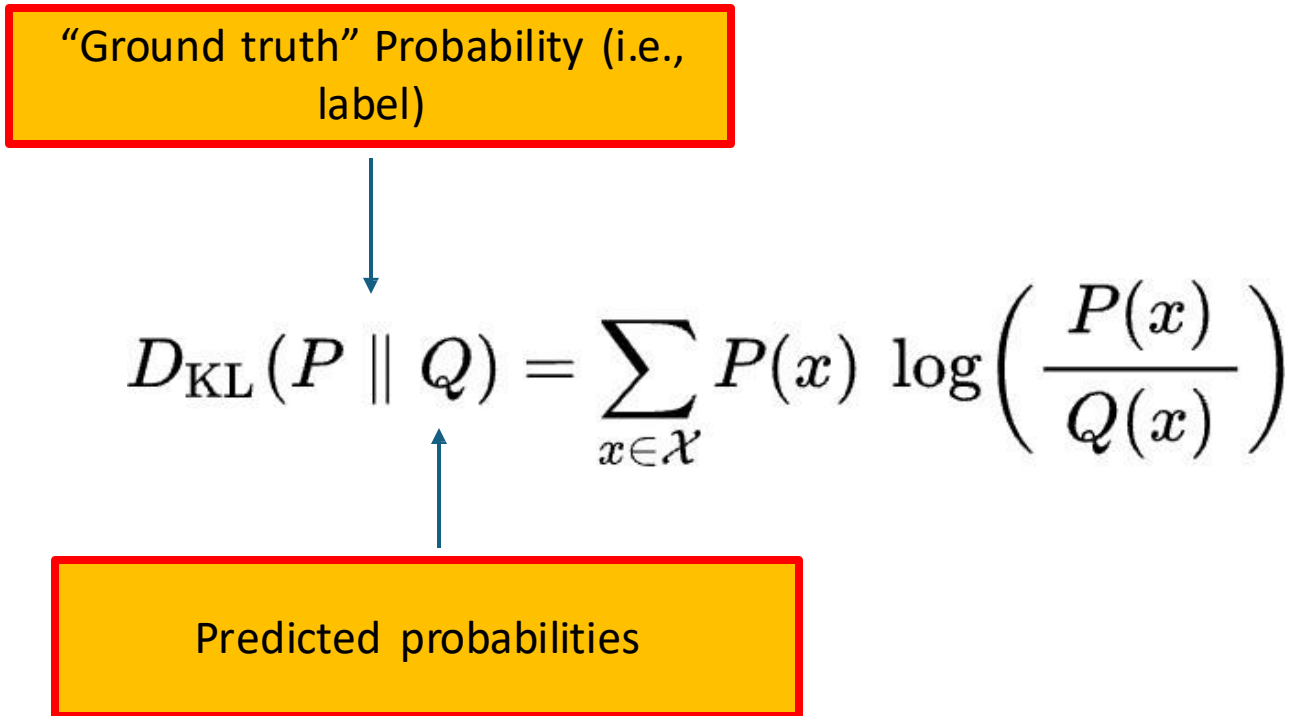
- One type of statistical distance
  - Distance between two probability distributions

"Ground truth" Probability (i.e., label)

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

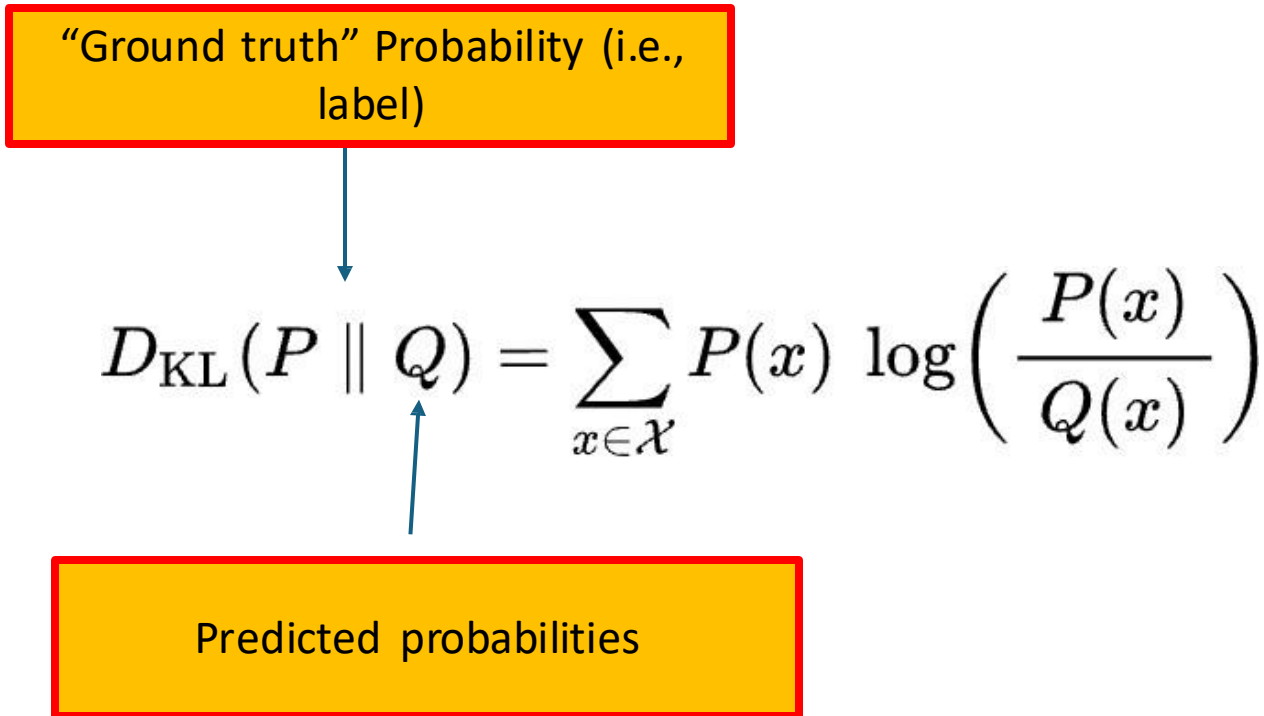
Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions



Kullback–Leibler divergence

- One type of statistical distance
  - Distance between two probability distributions



Binary Cross Entropy

KL Divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Cross Entropy (CE)

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

Binary Cross Entropy

KL Divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Cross Entropy (CE)

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

"Categorical Cross Entropy"

Binary Cross Entropy

KL Divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Cross Entropy (CE)

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

”Categorical Cross Entropy”

For Binary problems “Binary Cross Entropy” (BCE)

Cross Entropy Examples

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

Random choice between two categories (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.5, 0.5]$$

$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0] \cdot \log([0.5, 0.5]) = 0.693$$

Cross Entropy Examples

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

Random choice between two categories (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.5, 0.5]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0] \cdot \log([0.5, 0.5]) = 0.693$$

Random choice between 10 categories (one sample):

$$\mathrm{y} = [1, 0, \dots], \hat{\mathrm{y}} = [0.1, 0.1, \dots]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0] \cdot \log([0.1, 0.1, \dots]) = 2.3$$

Cross Entropy Examples

$$CE(y, \hat{y}) = -\sum_{i}^{n} y_i \log \hat{y}_i$$

Random choice between two categories (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.5, 0.5]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0] \cdot \log([0.5, 0.5]) = 0.693$$

Random choice between 10 categories (one sample):

$$\mathrm{y} = [1, 0, \dots], \hat{\mathrm{y}} = [0.1, 0.1, \dots]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0] \cdot \log([0.1, 0.1, \dots]) = 2.3$$

Random choice between 100 categories (one sample):

$$\mathrm{y} = [1, 0, \dots], \hat{\mathrm{y}} = [0.01, 0.01, \dots]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0, \dots] \cdot \log([0.01, 0.01, \dots]) = 4.6$$

Cross Entropy Examples

Random choice between two categories (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.5, 0.5]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0] \cdot \log([0.5, 0.5]) = 0.693$$

True class is higher output (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.75, 0.25]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = ???$$

True class is lower output (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.25, 0.75]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = ???$$

Cross Entropy Examples

Random choice between two categories (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.5, 0.5]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0] \cdot \log([0.5, 0.5]) = 0.693$$

True class is higher output (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.75, 0.25]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = ???$$

0.28

True class is lower output (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.25, 0.75]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = ???$$

Cross Entropy Examples

Random choice between two categories (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.5, 0.5]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = -[1, 0] \cdot \log([0.5, 0.5]) = 0.693$$

True class is higher output (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.75, 0.25]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = ???$$

0.28

True class is lower output (one sample):

$$\mathrm{y} = [1, 0], \hat{\mathrm{y}} = [0.25, 0.75]$$
$$\mathrm{CE}(\mathrm{y}, \hat{\mathrm{y}}) = ???$$

1.3

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}} \sum_{i}^{n} y_i \log \hat{y}_i$$

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}} \sum_{i}^{n} y_i \log \hat{y}_i$$

What is this? (vector, scalar, matrix)

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}} \sum_{i}^{n} y_i \log \hat{y}_i$$

What is this? (vector, scalar, matrix)

What is this? (vector, scalar, matrix)

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}} \sum_{i}^{n} y_i \log \hat{y}_i$$

What is this? (vector, scalar, matrix)

What is this? (vector, scalar, matrix)

What is this? (vector, scalar, matrix)

Derivative of Cross Entropy

$$\frac{dL}{d\hat{y}} = -\frac{d}{d\hat{y}} \sum_{i}^{n} y_i \log \hat{y}_i$$

$$\frac{dL}{d\hat{y}} = -\sum_{i}^{n} -\frac{1}{p_i}$$

Probability of predicting **correct** label for example i

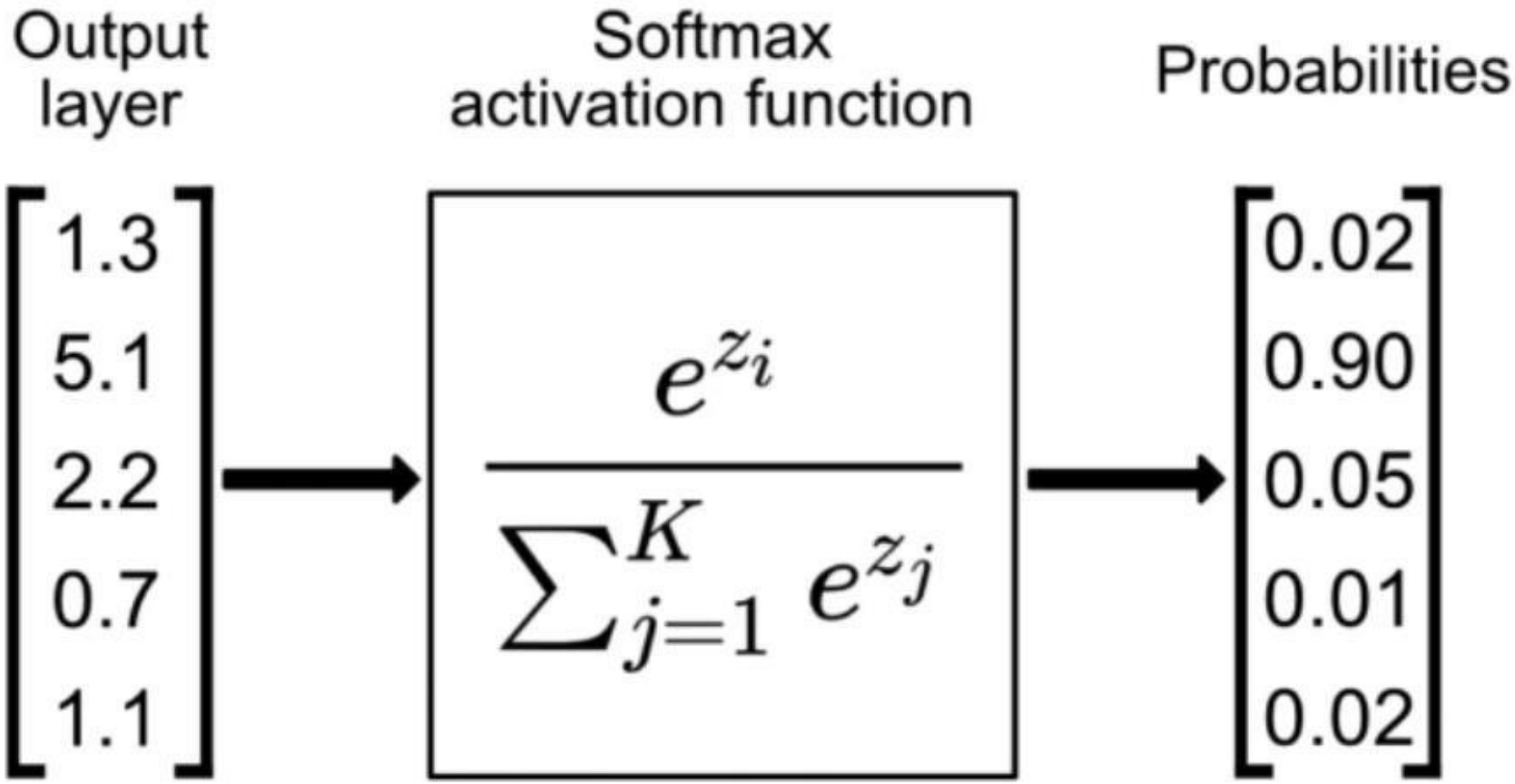
Probabilities

- If we have probabilities, we can use Cross Entropy
- How do we get probabilities?

Option #1: Normalize outputs (i.e., divide by their total)

Option #2: Use another function (i.e., softmax)

Softmax Function



Source: https://www.singlestore.com/blog/a-guide-to-softmax-activation-function/

What's the difference?

Consider a neural network with 2 outputs.

For one image, the network outputs [1, 2]. For a second image, the network outputs [10, 20].

What will be the predicted probabilities with normalization?

What's the difference?

Consider a neural network with 2 outputs.

For one image, the network outputs [1, 2]. For a second image, the network outputs [10, 20].

What will be the predicted probabilities with normalization?

[1/3, 2/3] for both examples

What's the difference?

Consider a neural network with 2 outputs.

For one image, the network outputs [1, 2]. For a second image, the network outputs [10, 20].

What will be the predicted probabilities with Softmax?

What's the difference?

Consider a neural network with 2 outputs.

For one image, the network outputs [1, 2]. For a second image, the network outputs [10, 20].

What will be the predicted probabilities with Softmax?

[0.26, 0.73] for [1, 2]
[0.00005, 0.99995] for [10, 20]

What's the difference?

Add 10 to each output

Consider a neural network with 2 outputs.

For one image, the network outputs [11, 12]. For a second image, the network outputs [20, 30].

What will be the predicted probabilities with Normalization?

What's the difference?

Add 10 to each output

Consider a neural network with 2 outputs.

For one image, the network outputs [11, 12]. For a second image, the network outputs [20, 30].

What will be the predicted probabilities with Normalization?

[0.47, 0.53] for [11, 12]
[0.4, 0.6] for [20, 30]

What's the difference?

Add 10 to each output

Consider a neural network with 2 outputs.

For one image, the network outputs [11, 12]. For a second image, the network outputs [20, 30].

What will be the predicted probabilities with Softmax?

What's the difference?

Add 10 to each output

Consider a neural network with 2 outputs.

For one image, the network outputs [11, 12]. For a second image, the network outputs [20, 30].

What will be the predicted probabilities with Softmax?

[0.26, 0.73] for [11, 12]
[0.00005, 0.99995] for [20, 30]
Exactly the same as [1, 2] and [10, 20]

What’s the difference?

What’s the difference?

Normalization is sensitive to additive changes, but not multiplicative changes

What's the difference?

Normalization is sensitive to additive changes, but not multiplicative changes

Softmax is sensitive to multiplicative changes, but not additive

What’s the difference?

- Normalization is sensitive to additive changes, but not multiplicative changes

- Softmax is sensitive to multiplicative changes, but not additive

Softmax also has other advantages:

What's the difference?

- Normalization is sensitive to additive changes, but not multiplicative changes

- Softmax is sensitive to multiplicative changes, but not additive

Softmax also has other advantages:

- - Tends to handle smaller probabilities better (less float underflow)

What's the difference?

Normalization is sensitive to additive changes, but not multiplicative changes

Softmax is sensitive to multiplicative changes, but not additive

Softmax also has other advantages:

- Tends to handle smaller probabilities better (less float underflow)
- Remember that log in our loss function? Remember the e^z in softmax? Our loss function becomes ~linear for our neuron outputs z

What's the difference?

Normalization is sensitive to additive changes, but not multiplicative changes

Softmax is sensitive to multiplicative changes, but not additive

Softmax also has other advantages:

- Tends to handle smaller probabilities better (less float underflow)
- Remember that log in our loss function? Remember the e^z in softmax? Our loss function becomes ~linear for our neuron outputs z
- Maybe has issues with overflow… (outputs can become inf or NaN)

Derivative of Softmax

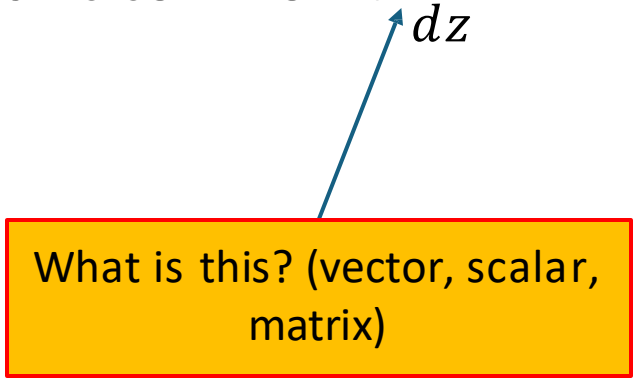
$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Want to know: $\frac{da}{dz}$

Derivative of Softmax

$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Want to know: $\frac{da}{dz}$



Derivative of Softmax

$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Want to know: $\frac{da}{dz}$

a and *z* are both vectors, therefore $\frac{da}{dz}$ *is a Jacobian matrix*

Derivative of Softmax

$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Want to know: $\frac{da}{dz}$

What is $\frac{\partial a_i}{\partial z_j}$?

Derivative of Softmax

$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Quotient rule!

Want to know: $\frac{da}{dz}$

What is $\frac{\partial a_i}{\partial z_j}$?

Derivative of Softmax

$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Want to know: $\frac{da}{dz}$

What is $\frac{\partial a_i}{\partial z_j}$?

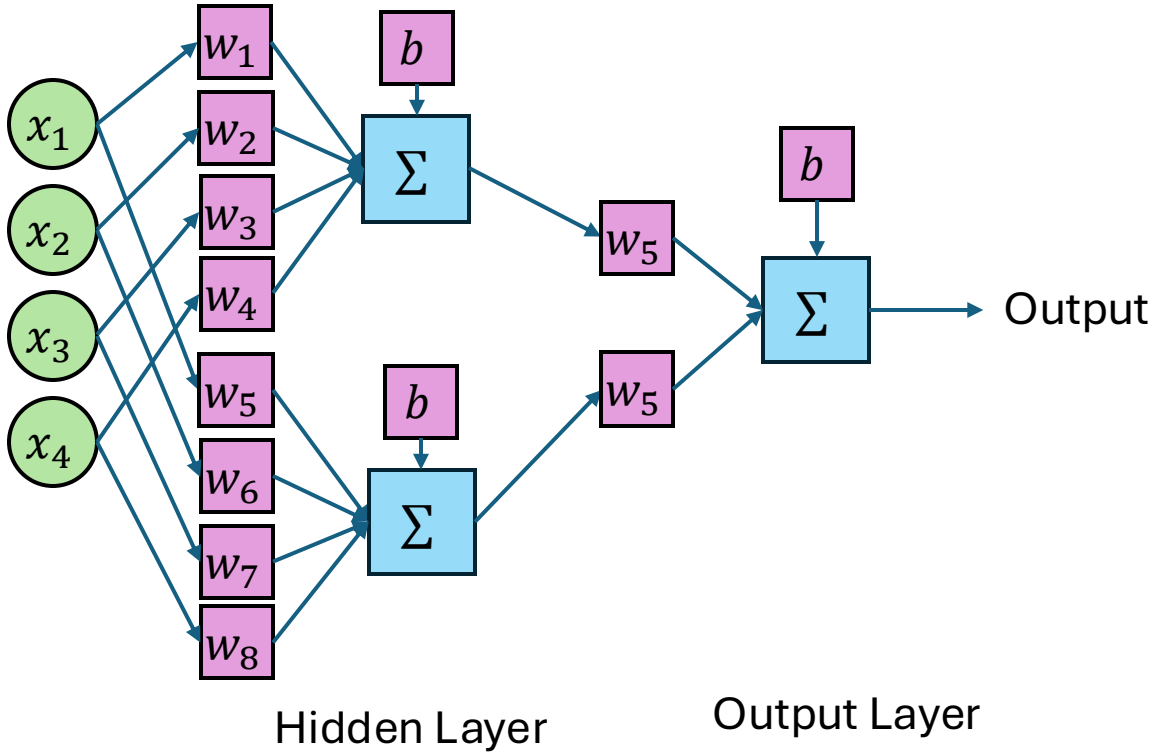
If $i == j$, then $\frac{\partial a_i}{\partial z_i} = a_i \cdot (1 - a_i)$

If $i! = j$, then $\frac{\partial a_i}{\partial z_j} = -a_i \cdot a_j$

Hyperparameter Tuning

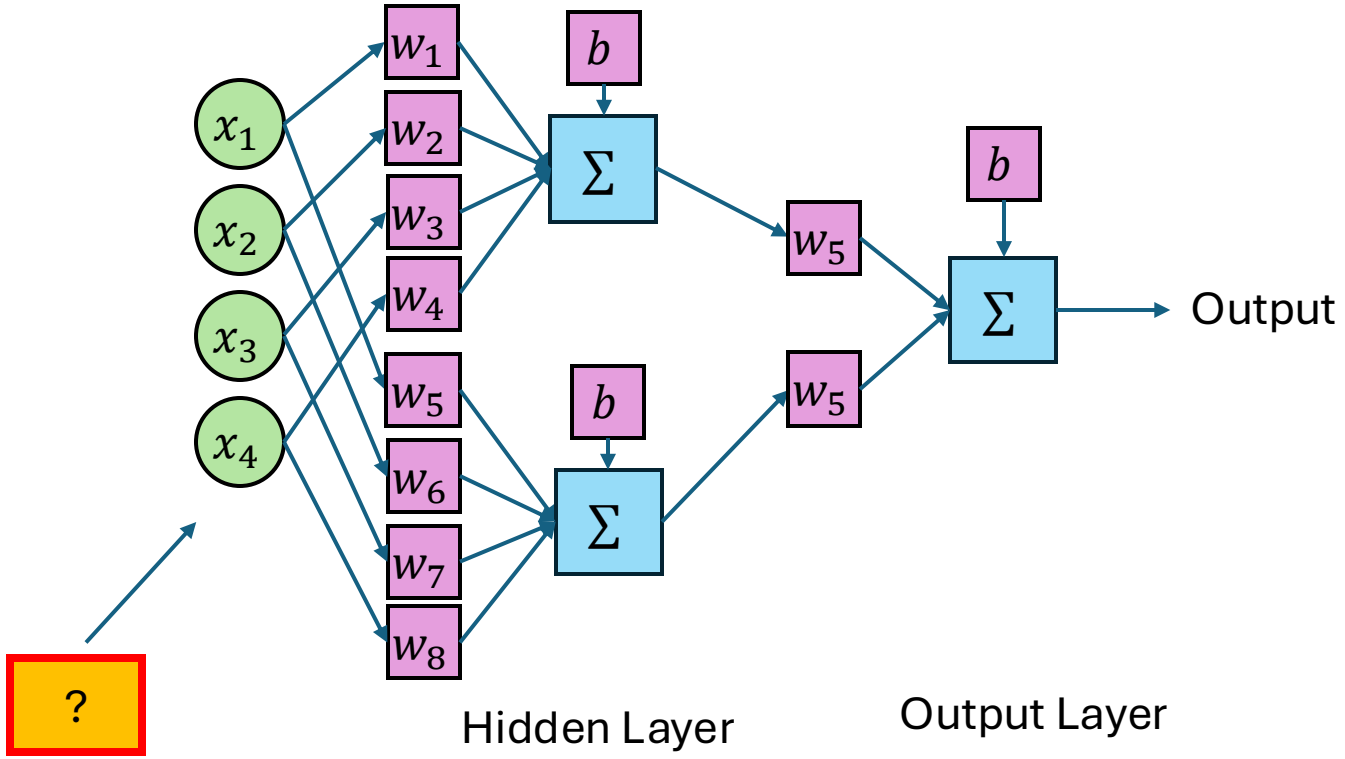
Hyperparameters

What do you (the programmer) have control of when training neural networks?



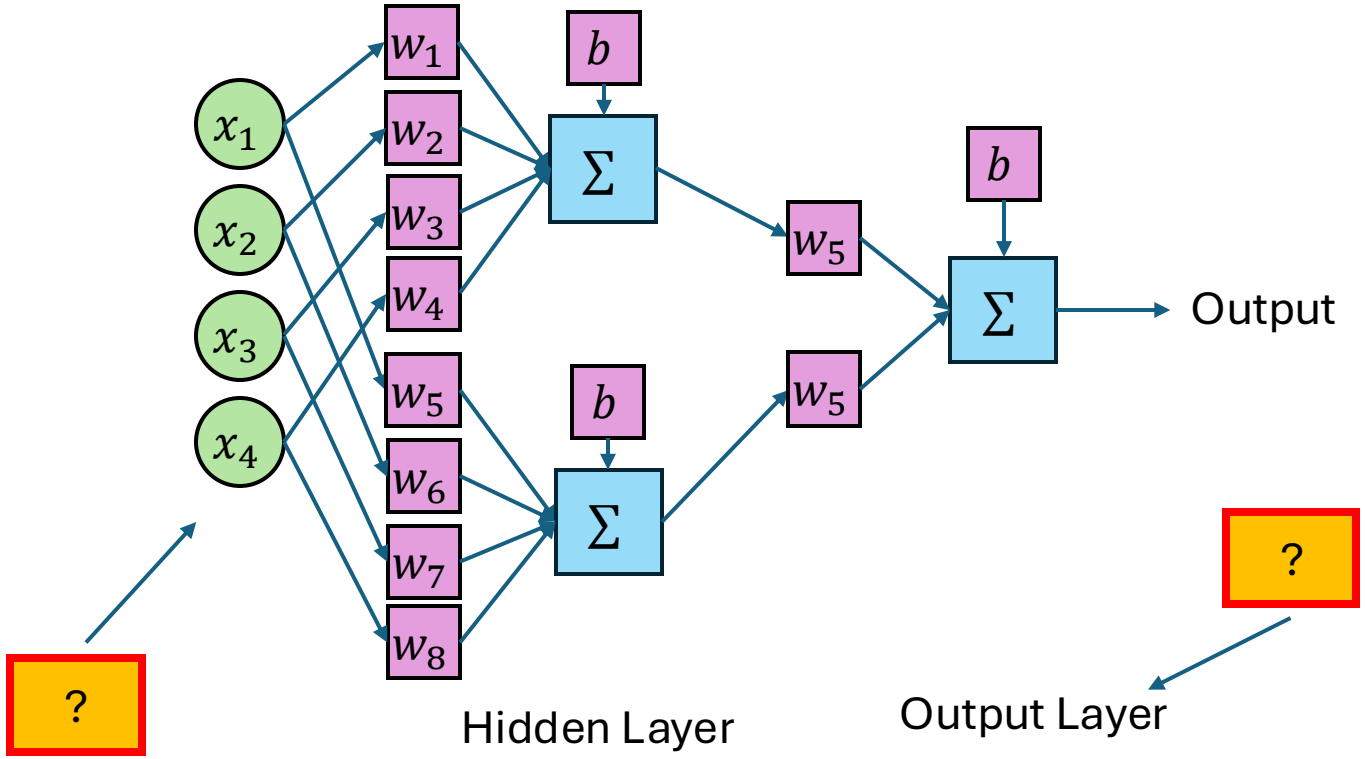
Hyperparameters

What do you (the programmer) have control of when training neural networks?



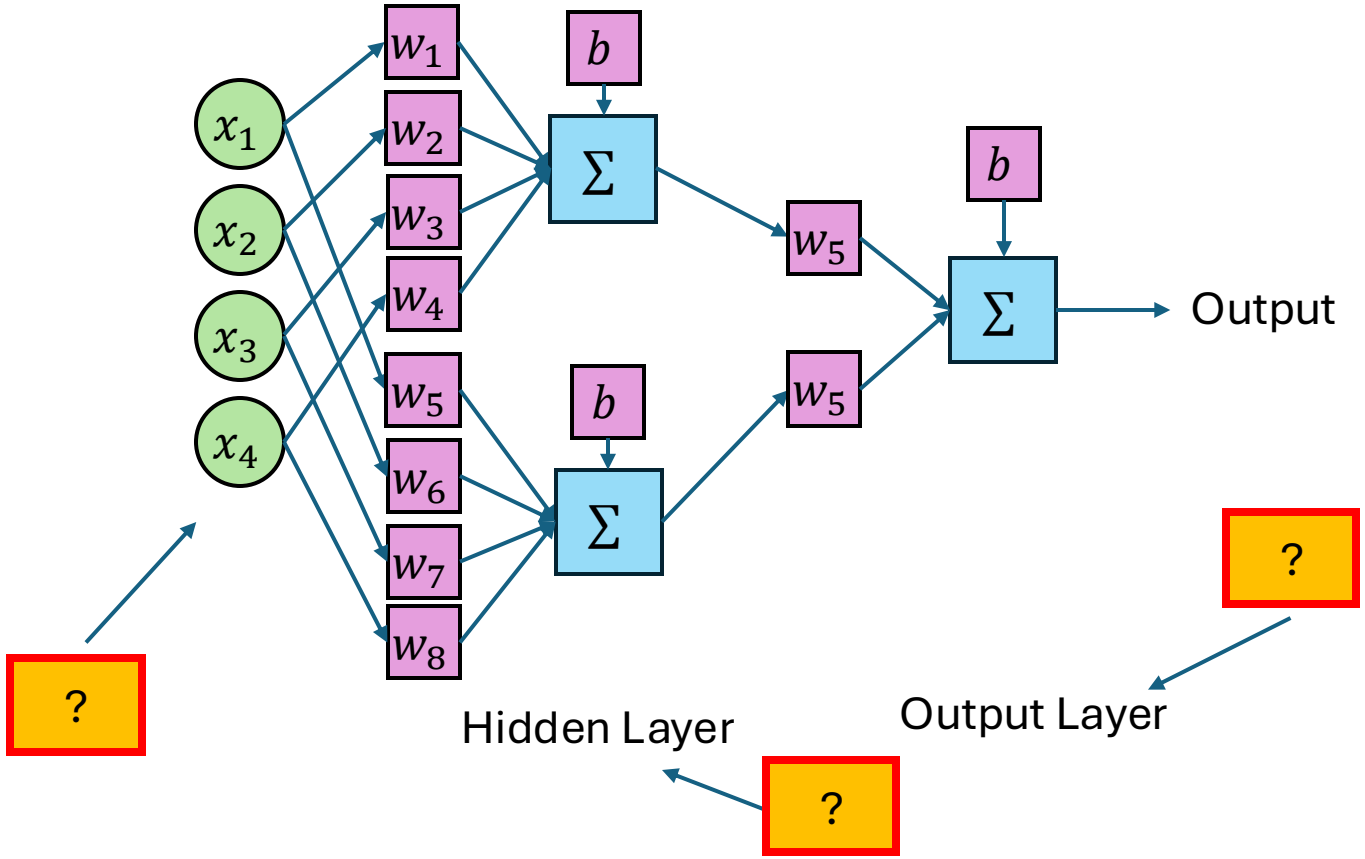
Hyperparameters

What do you (the programmer) have control of when training neural networks?



Hyperparameters

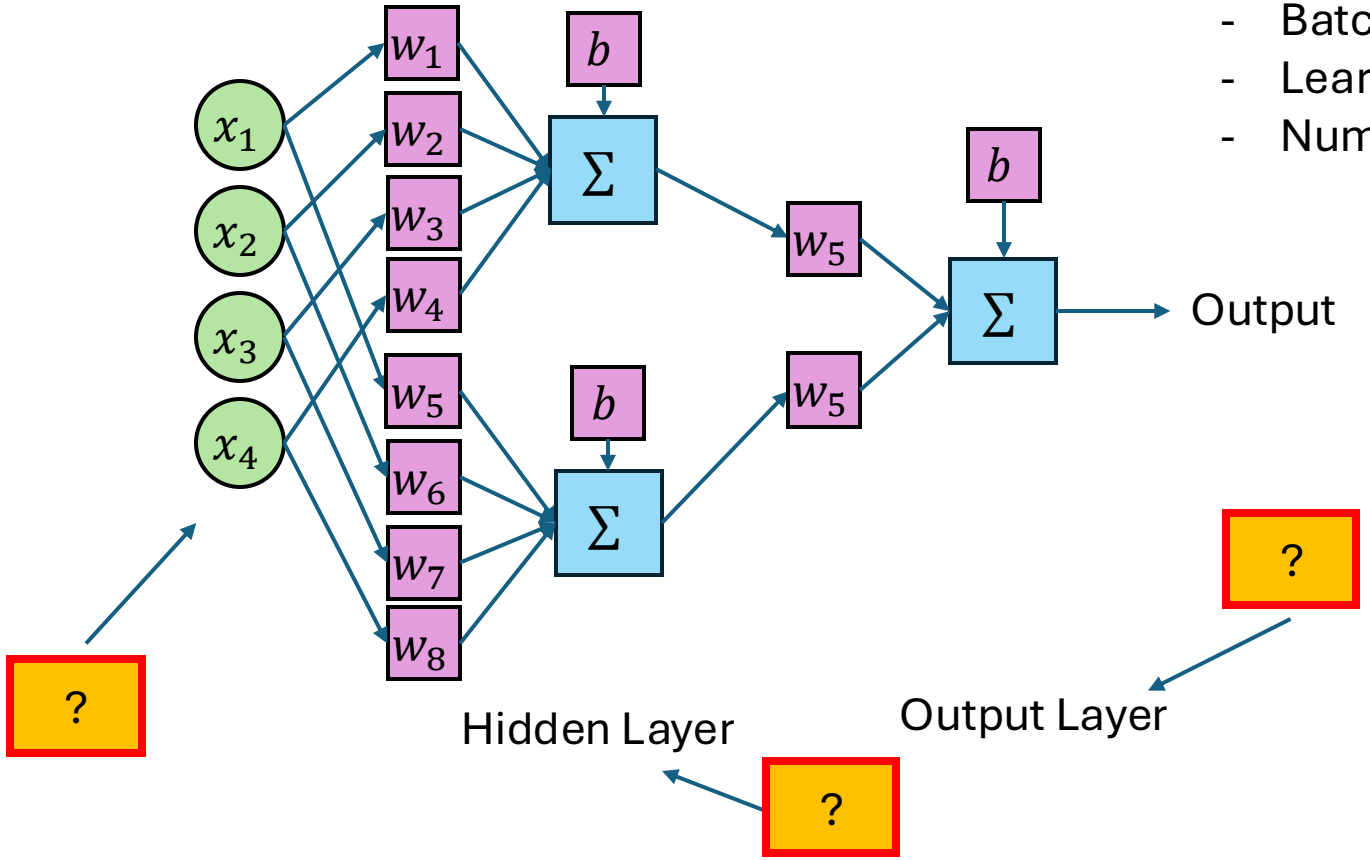
What do you (the programmer) have control of when training neural networks?



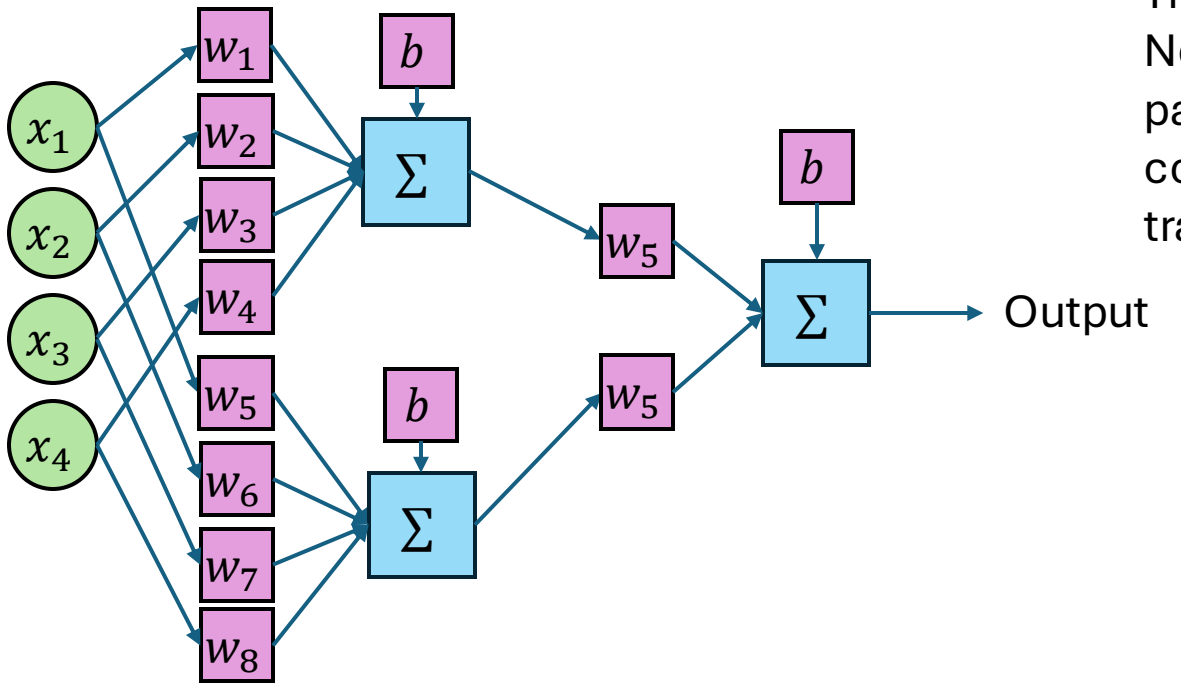
Hyperparameters

What do you (the programmer) have control of when training neural networks?

- Network Initialization
- Hidden Layer Size
- Number of hidden layers
- Activation Functions
- Optimizer (SGD, Adam, RMSProp)
- Batch Size
- Learning rate
- Number of Epochs



Hyperparameters

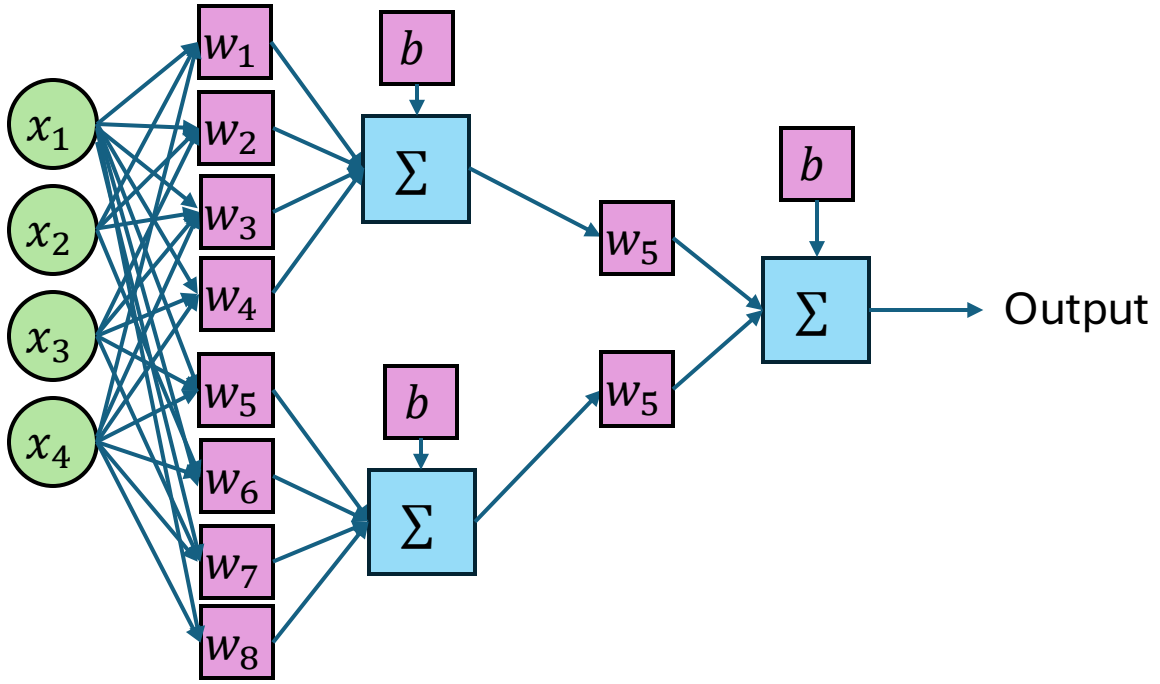


The **parameters** of a Neural Network are what is trained (e.g., weights and biases).

The **hyperparameters** of a Neural Network are the parameters that **you** have control of that control that training.

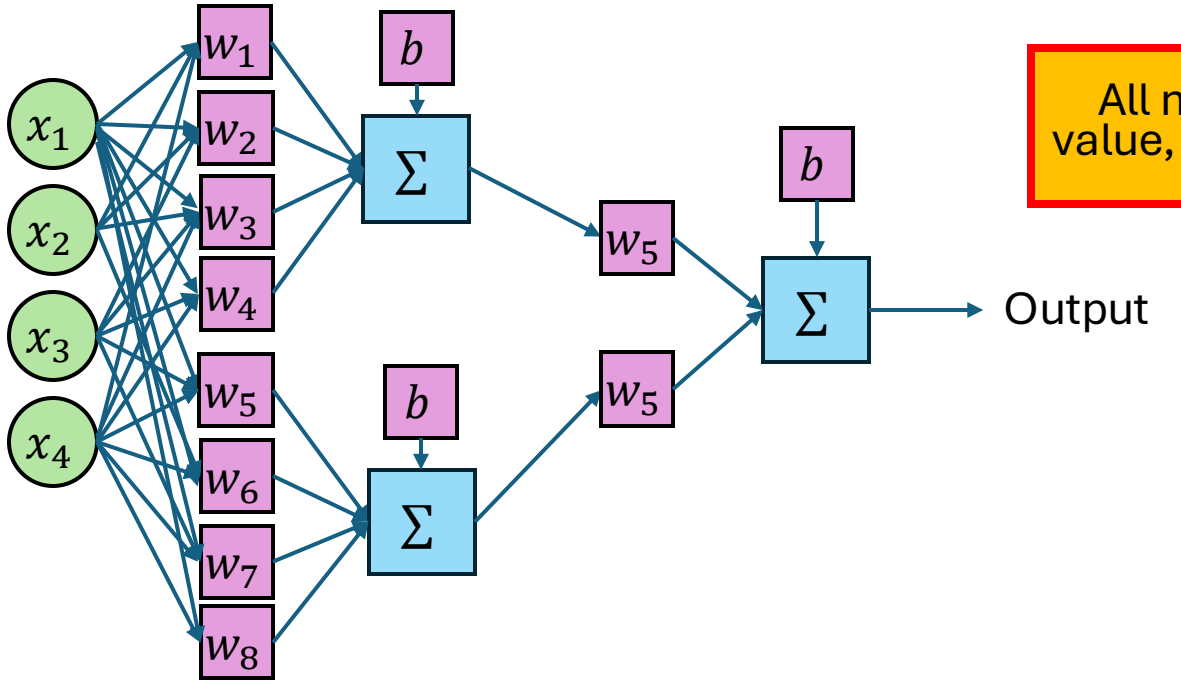
Network Initialization

What if we begin with all parameters set to 0?



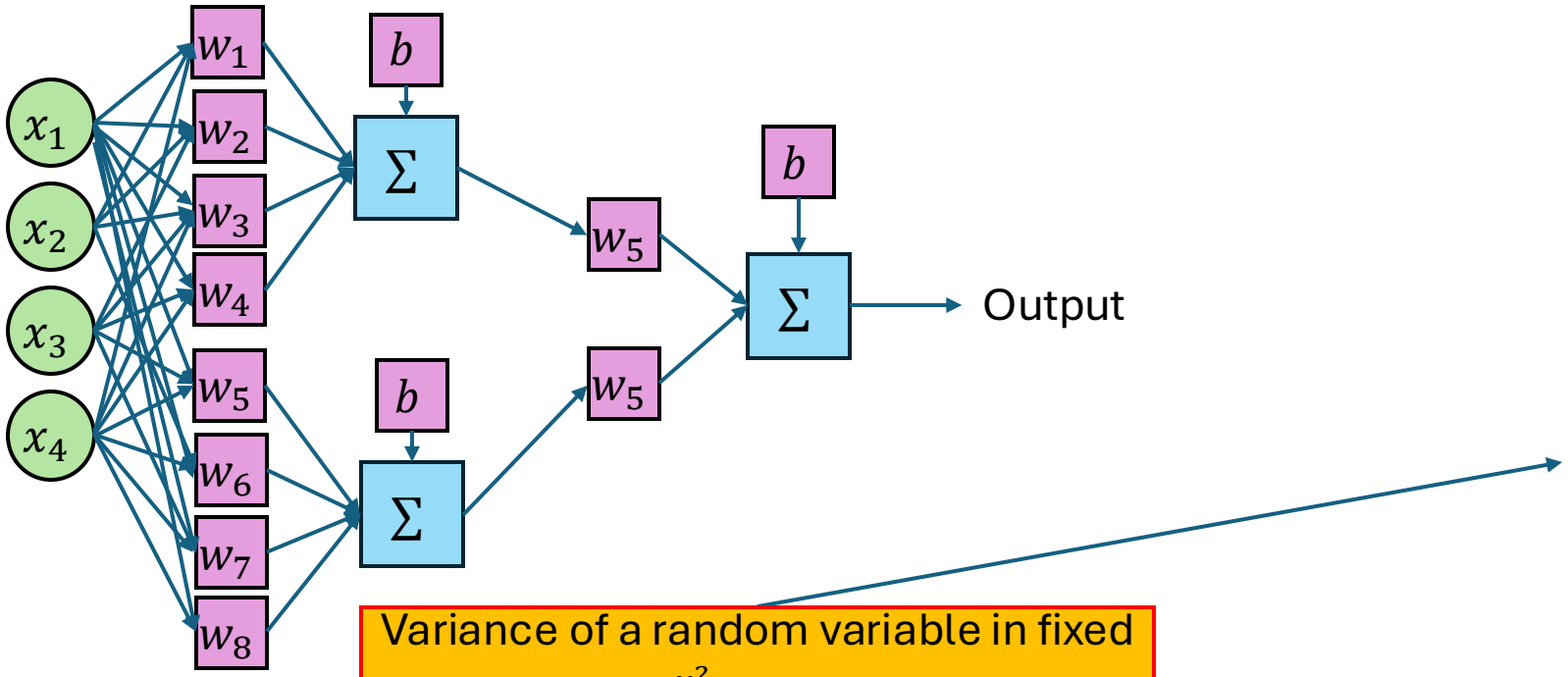
Network Initialization

What if we begin with all parameters set to 0?



All neurons would have the same value, **gradients would be the same.**

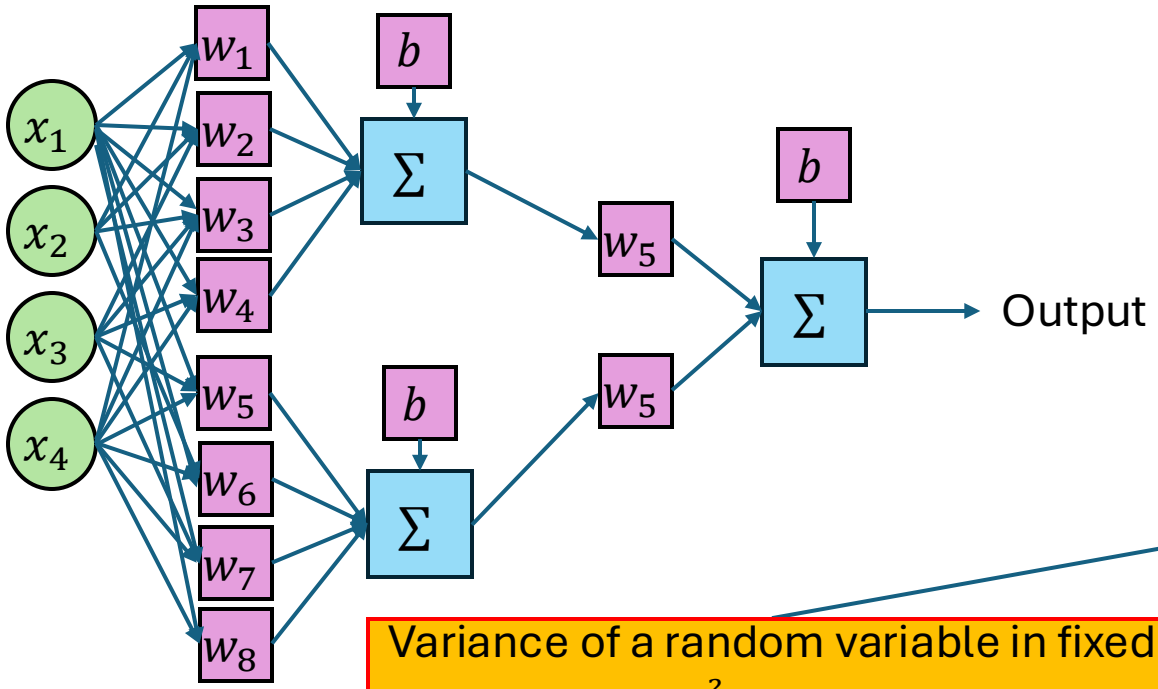
Network Initialization



Variance of a random variable in fixed range [-x, x] is $\frac{x^2}{3}$ (easy to derive from definition of variance)

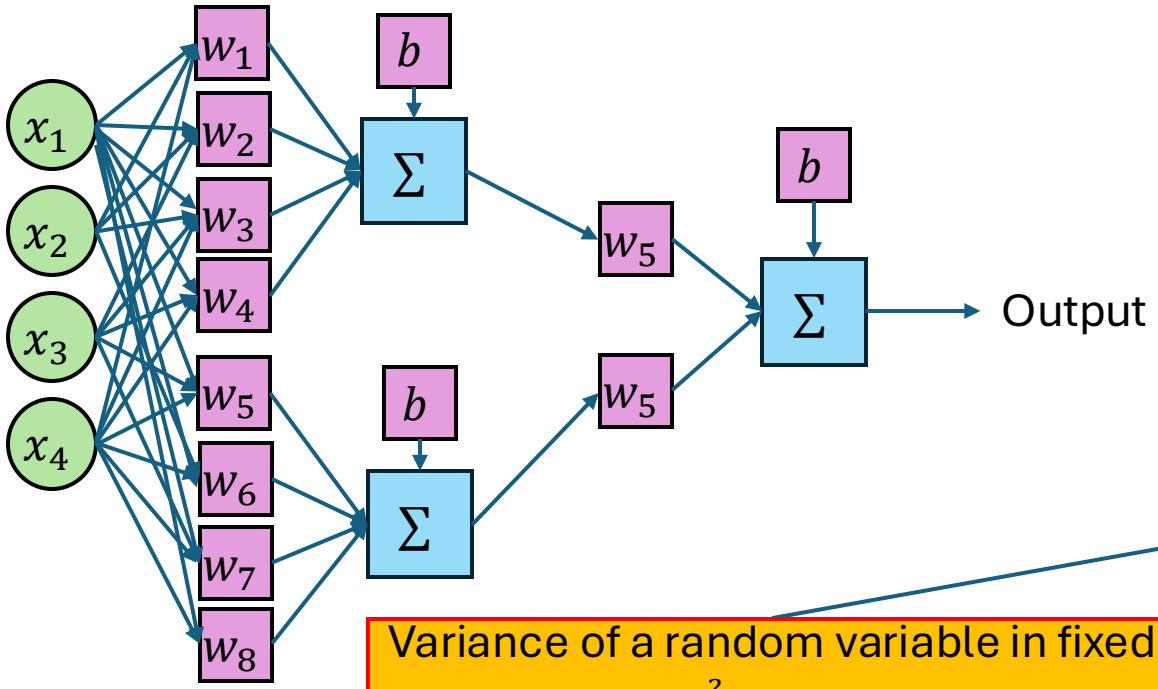
Network Initialization

Idea #1: Uniform random weights between -1 and 1 (works fine)



Variance of a random variable in fixed range [-x, x] is $\frac{x^2}{3}$ (easy to derive from definition of variance)

Network Initialization

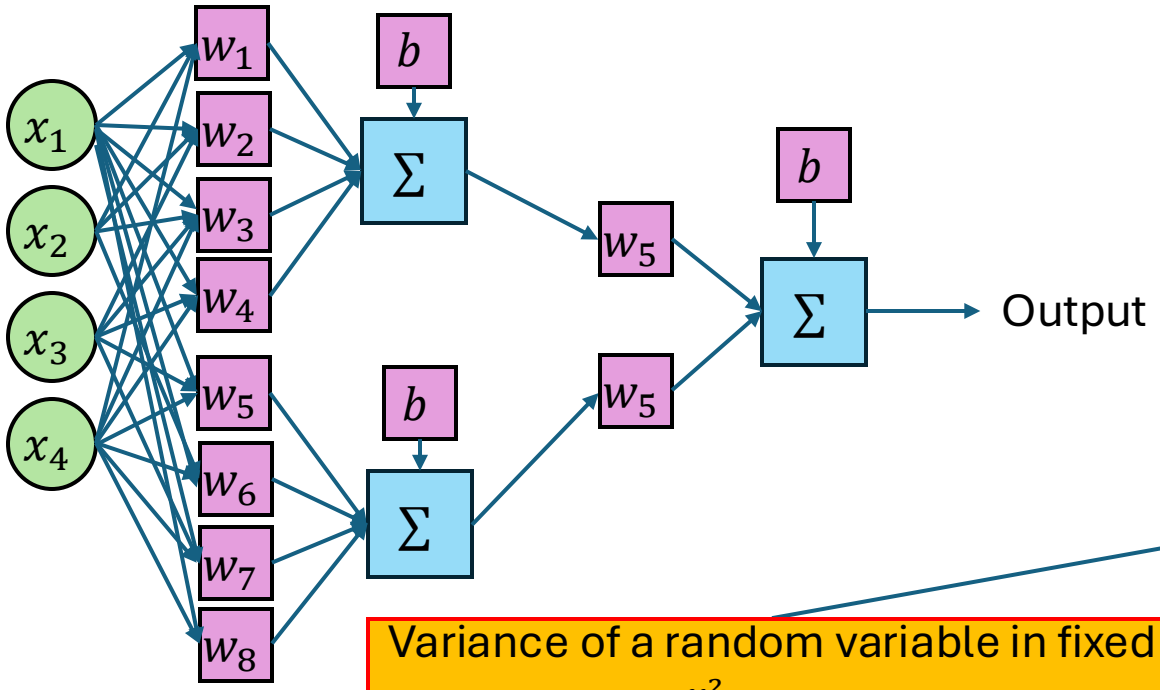


Idea #1: Uniform random weights between -1 and 1 (works fine)

But… what if there are many many weights in a layer? The scale of the output can grow.

Variance of a random variable in fixed range [-x, x] is $\frac{x^2}{3}$ (easy to derive from definition of variance)

Network Initialization



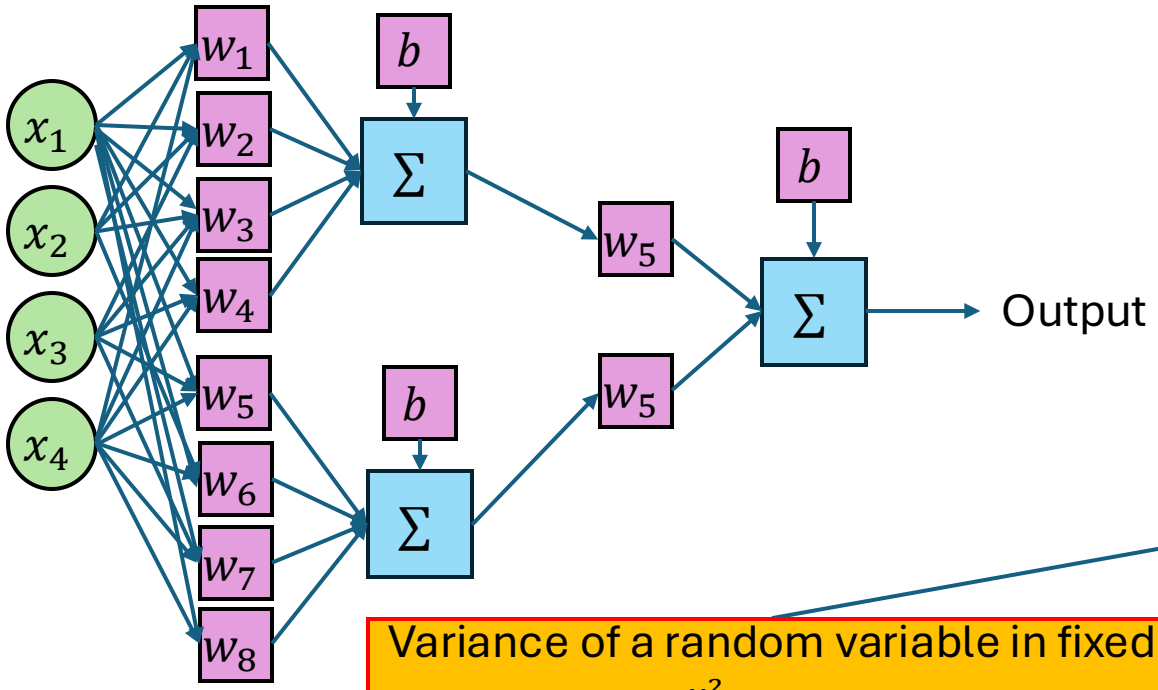
Idea #1: Uniform random weights between -1 and 1 (works fine)

But… what if there are many many weights in a layer? The scale of the output can grow.

Idea #2: Xavier (Glorot) initialization:

Variance of a random variable in fixed range [-x, x] is $\frac{x^2}{3}$ (easy to derive from definition of variance)

Network Initialization



Idea #1: Uniform random weights between -1 and 1 (works fine)

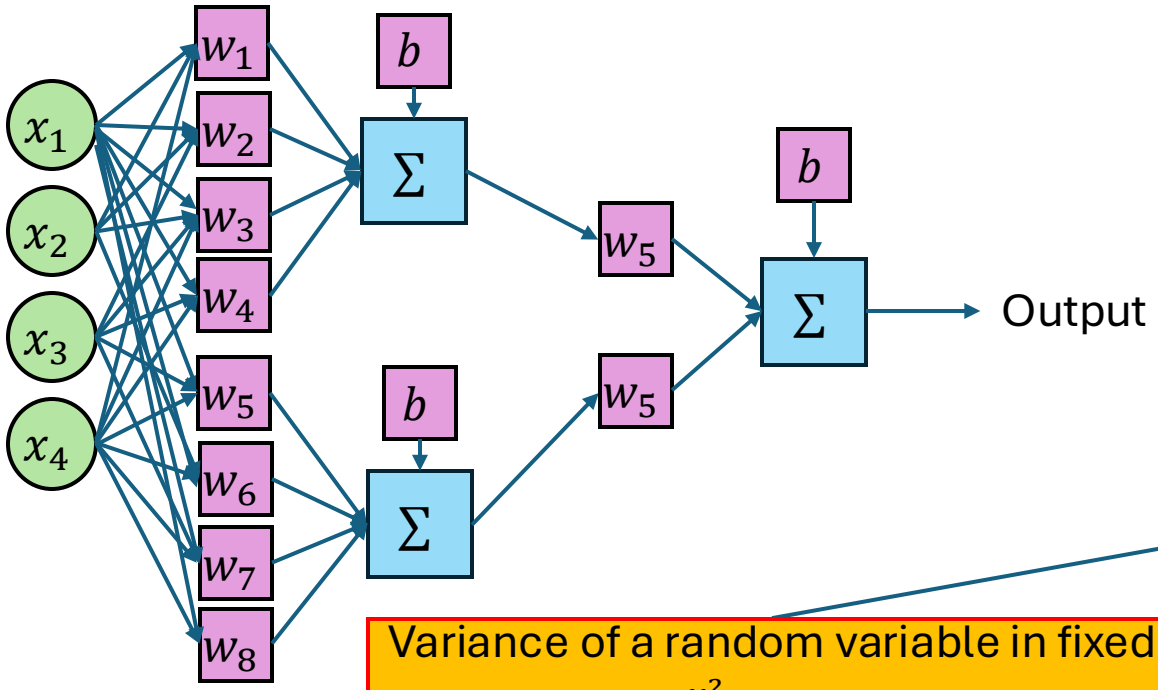
But… what if there are many many weights in a layer? The scale of the output can grow.

Idea #2: Xavier (Glorot) initialization:

Uniform: Initialize each weight uniformly at random in the range [-x, x] with $x = \sqrt{\frac{6}{n_{in}+n_{out}}}$

Variance of a random variable in fixed range [-x, x] is $\frac{x^2}{3}$ (easy to derive from definition of variance)

Network Initialization



Idea #1: Uniform random weights between -1 and 1 (works fine)

But… what if there are many many weights in a layer? The scale of the output can grow.

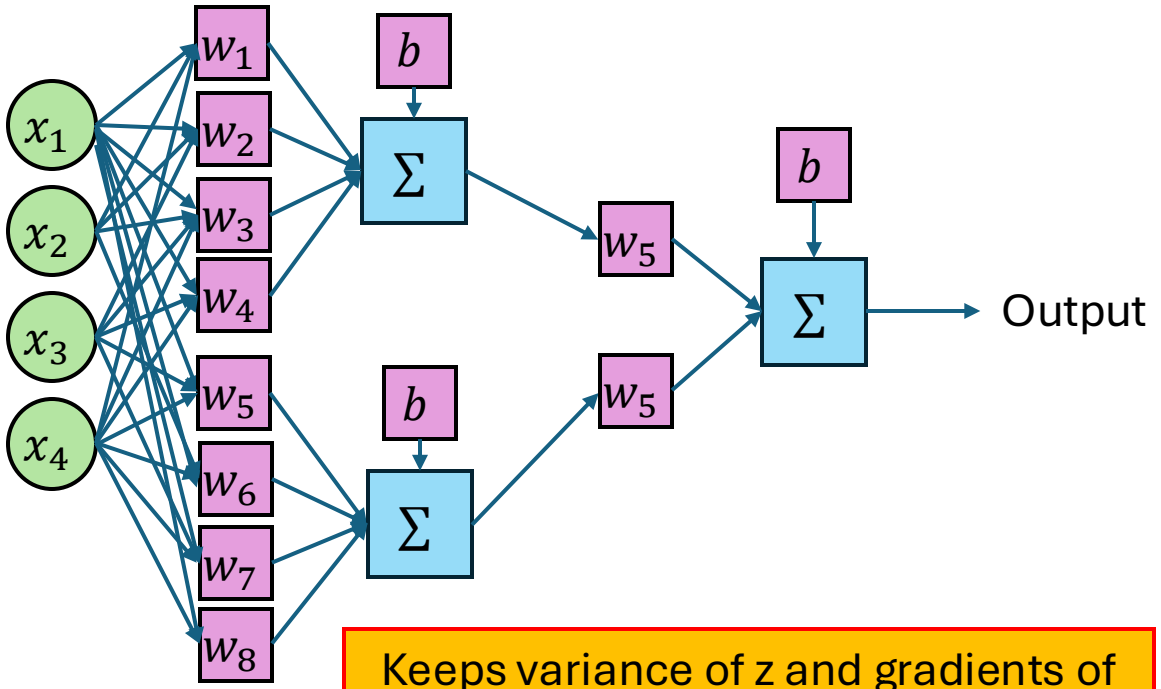
Idea #2: Xavier (Glorot) initialization:

Uniform: Initialize each weight uniformly at random in the range [-x, x] with $x = \sqrt{\frac{6}{n_{in}+n_{out}}}$

Normal: Initialize each weight with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{in}+n_{out}}}$

Variance of a random variable in fixed range [-x, x] is $\frac{x^2}{3}$ (easy to derive from definition of variance)

Network Initialization



Keeps variance of z and gradients of weights the same for each layer at initialization.

Idea #1: Uniform random weights between -1 and 1 (works fine)

But… what if there are many many weights in a layer? The scale of the output can grow.

Idea #2: Xavier (Glorot) initialization:

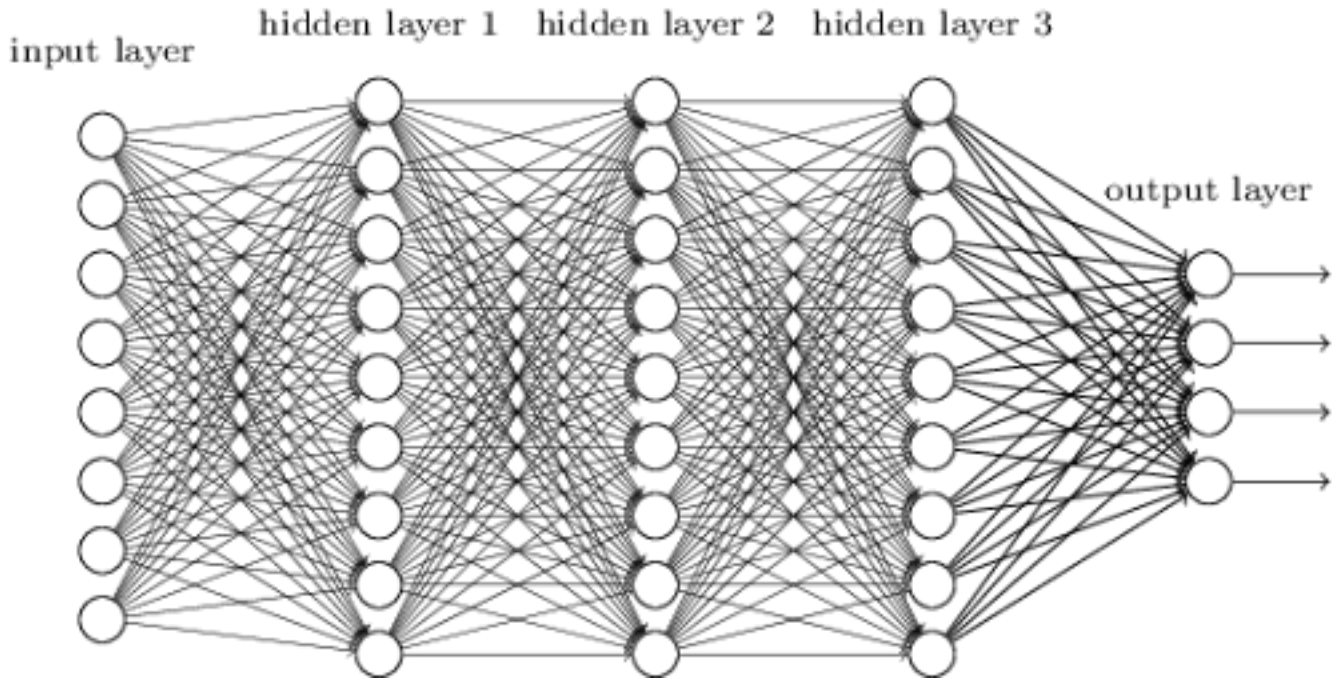
Uniform: Initialize each weight uniformly at random in the range [-x, x] with $x = \sqrt{\frac{6}{n_{in}+n_{out}}}$

Normal: Initialize each weight with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{in}+n_{out}}}$

Understanding the difficulty of training deep feedforward neural networks. ***Xavier Glorot, Yoshua Bengio***

Hidden Layers

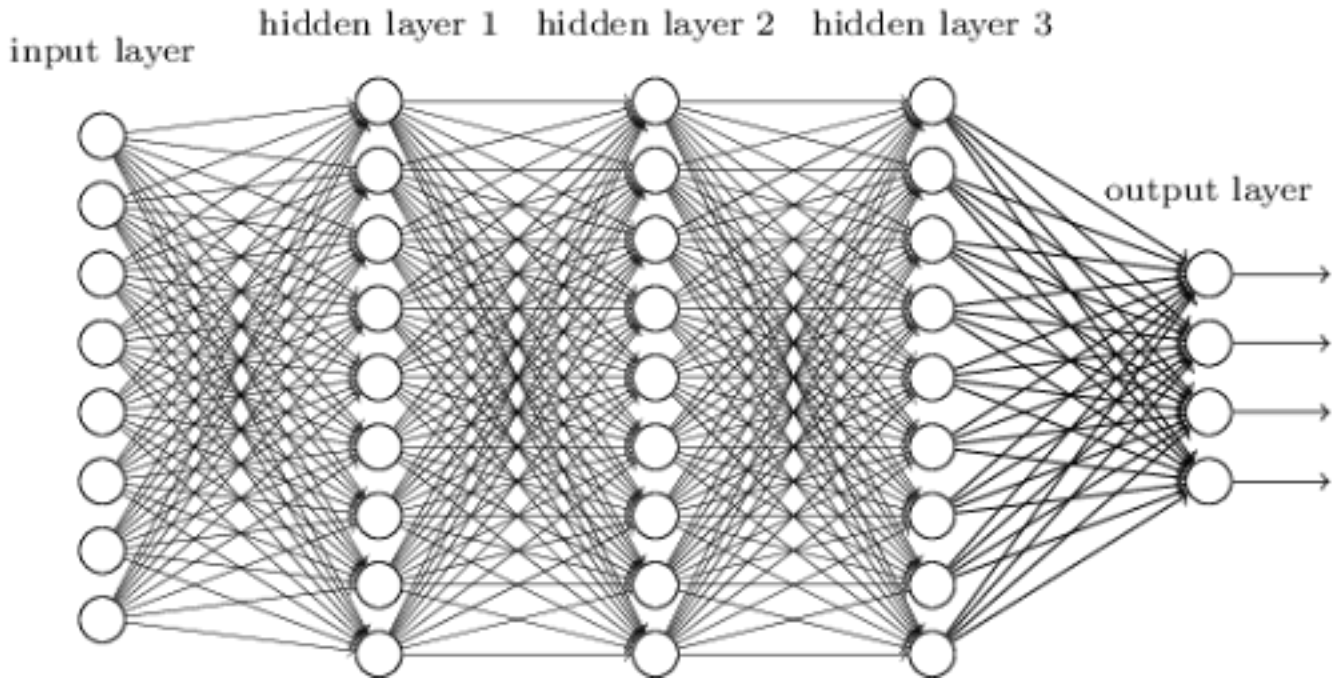
- How deep (# hidden layers) should your network be?
- How wide (# neurons in a layer) should your network be?



Hidden Layers

- How deep (# hidden layers) should your network be?
- How wide (# neurons in a layer) should your network be?

How complex is the problem you are trying to solve?

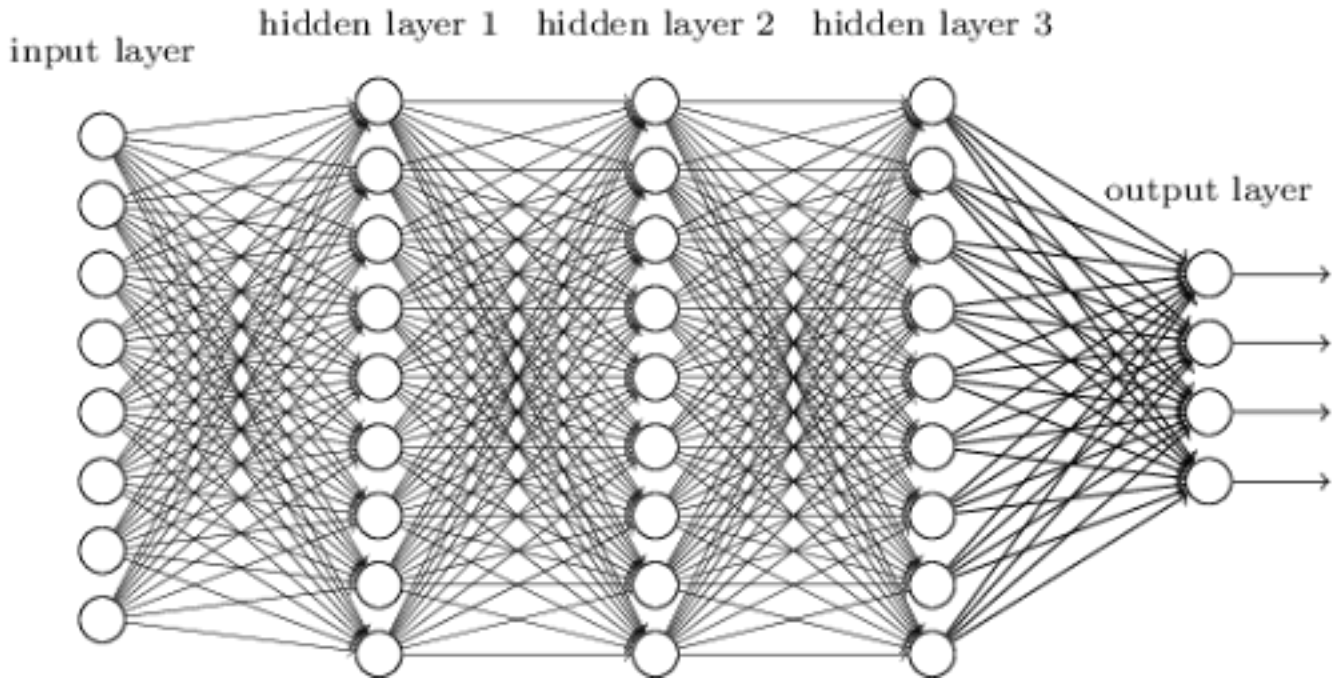


Hidden Layers

- How deep (# hidden layers) should your network be?
- How wide (# neurons in a layer) should your network be?

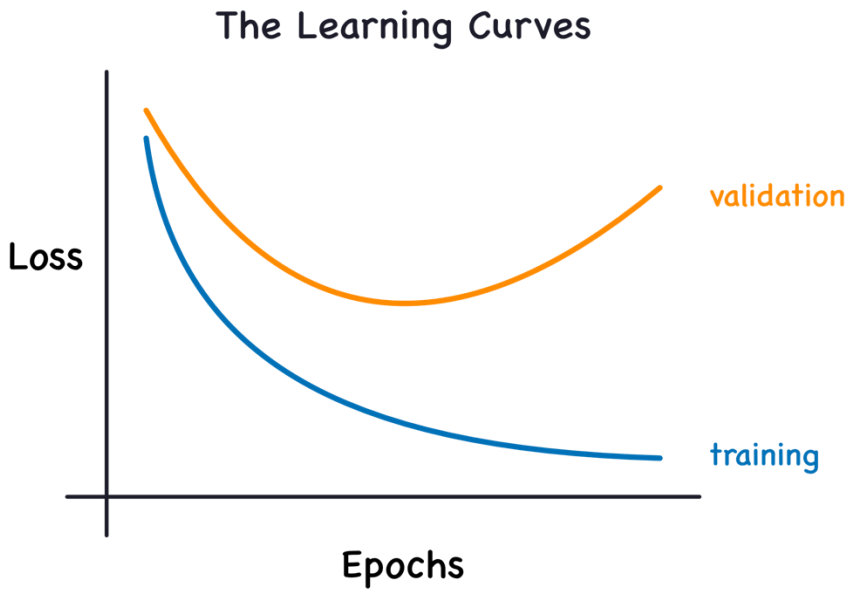
How complex is the problem you are trying to solve?

Process of (informed) trial and error. How do you know if one hyper-parameter setting is better than another?



How to use the Validation Set

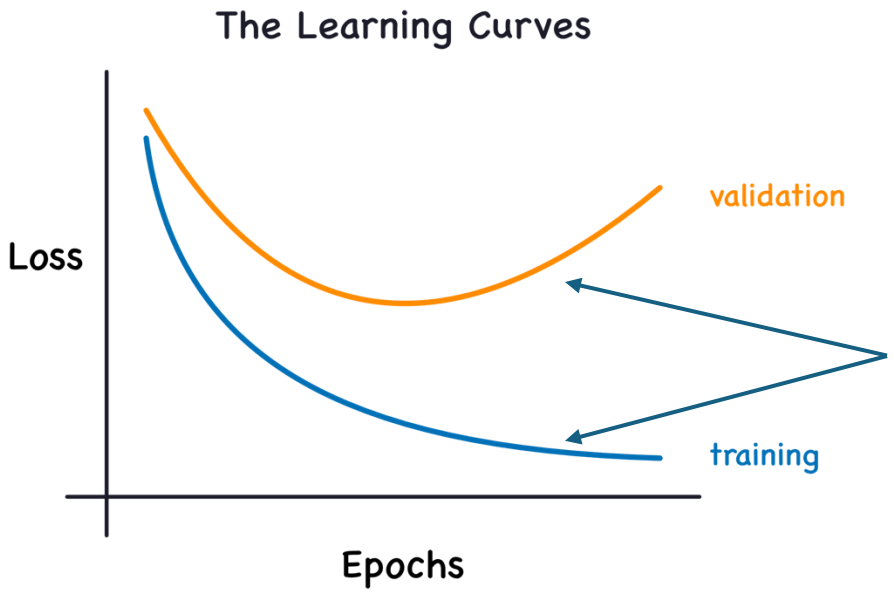
(In theory)



https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

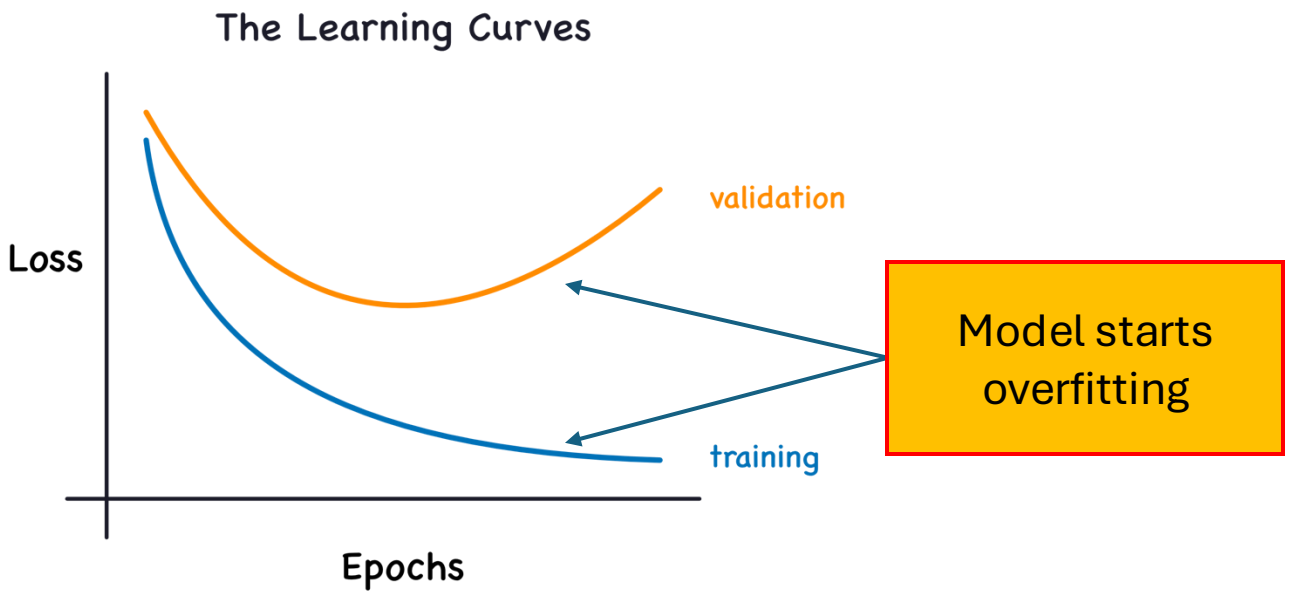
(In theory)



https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

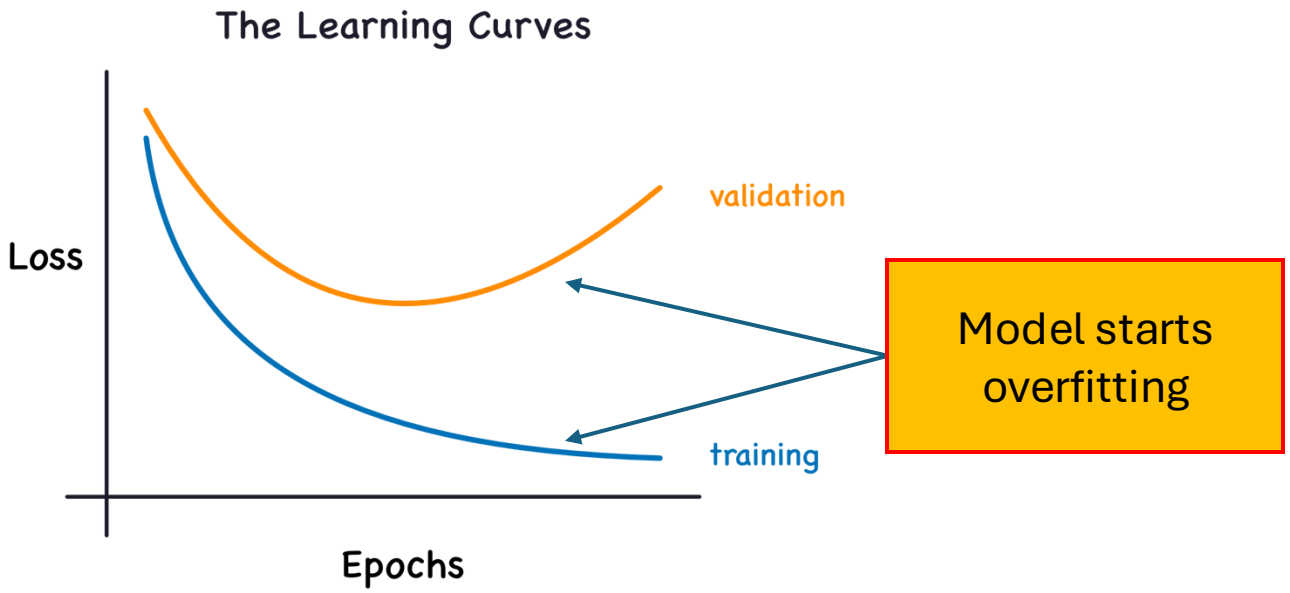
(In theory)



https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



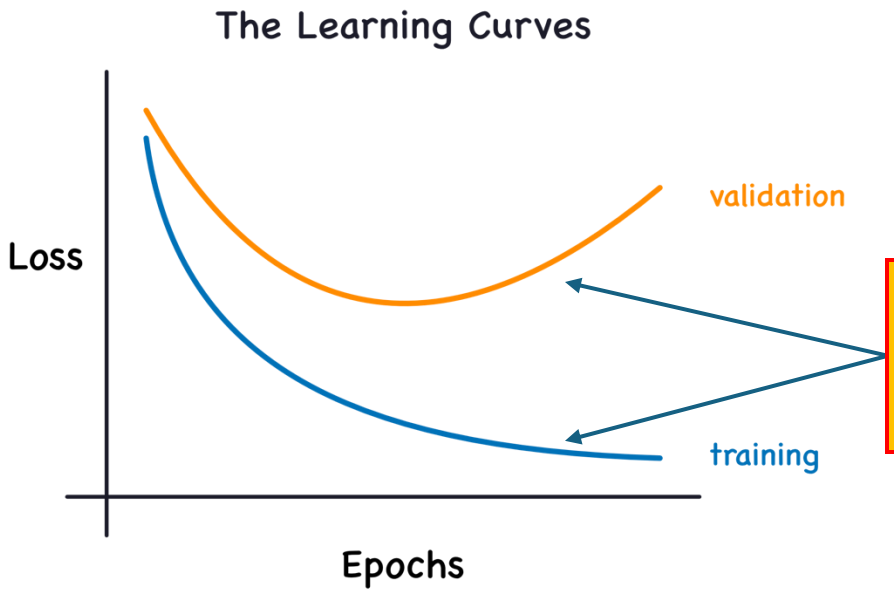
Early Stopping Algorithm

1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



Model starts overfitting

Early Stopping Algorithm

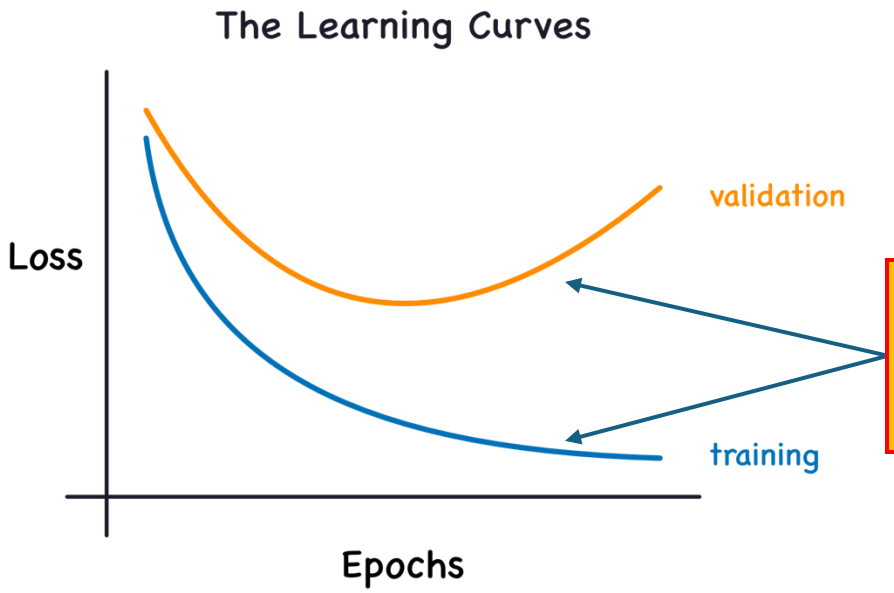
1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

What if your validation loss is much higher than training loss?

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



Early Stopping Algorithm

1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

Model starts overfitting

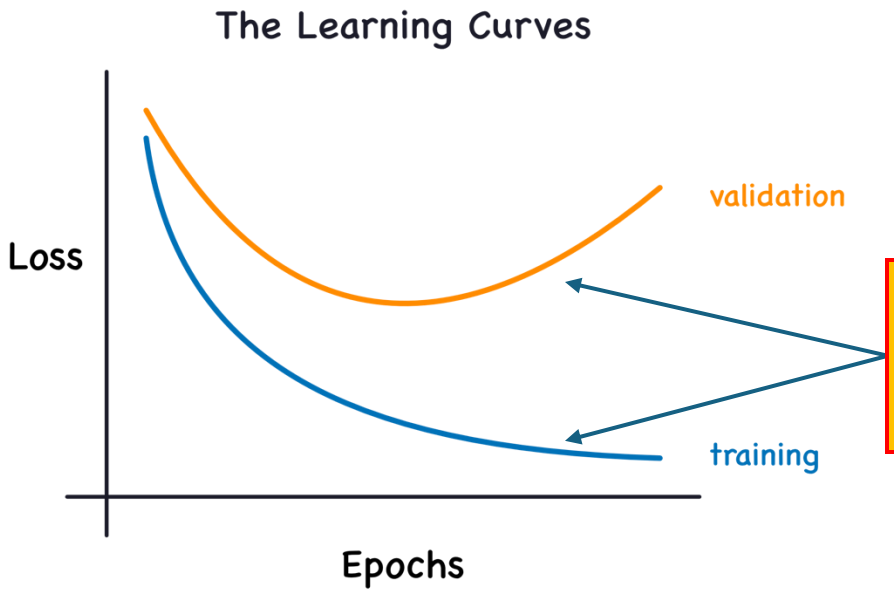
What if your validation loss is much higher than training loss?

Your model has overfit, try **reducing** its size

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



Model starts overfitting

Early Stopping Algorithm

1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

What if your validation loss is much higher than training loss?

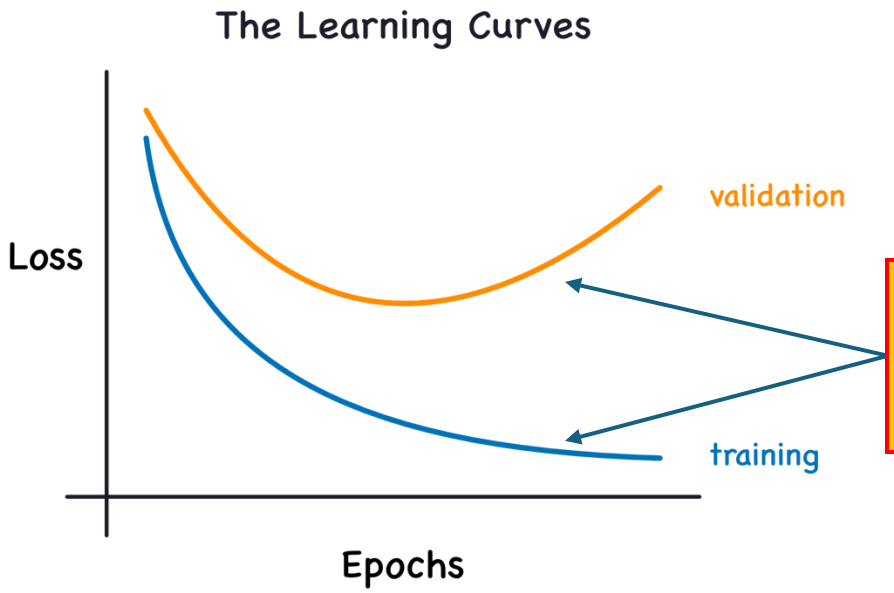
Your model has overfit, try **reducing** its size

What if your validation loss and training loss are both high?

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



Model starts overfitting

Early Stopping Algorithm

1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

What if your validation loss is much higher than training loss?

Your model has overfit, try **reducing** its size

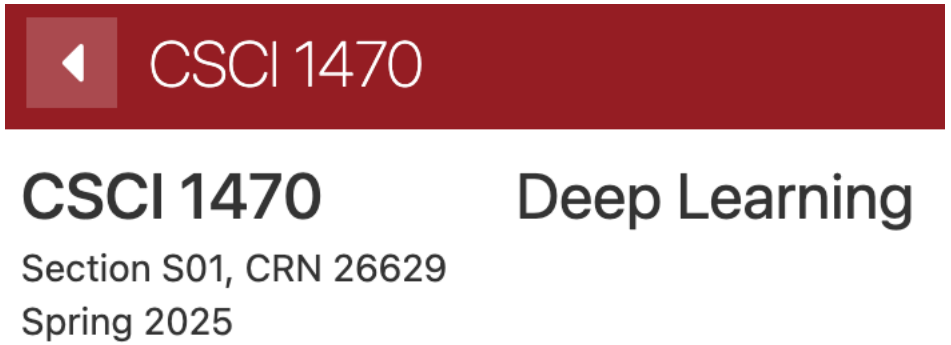
What if your validation loss and training loss are both high?

Your model has underfit, try **increasing** its size

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

Is adding more width or depth better?

Is adding more width or depth better?



Depth-Width Tradeoffs in Approximating Natural Functions with Neural Networks

Itay Safran
Weizmann Institute of Science
itay.safran@weizmann.ac.il

Ohad Shamir
Weizmann Institute of Science
ohad.shamir@weizmann.ac.il

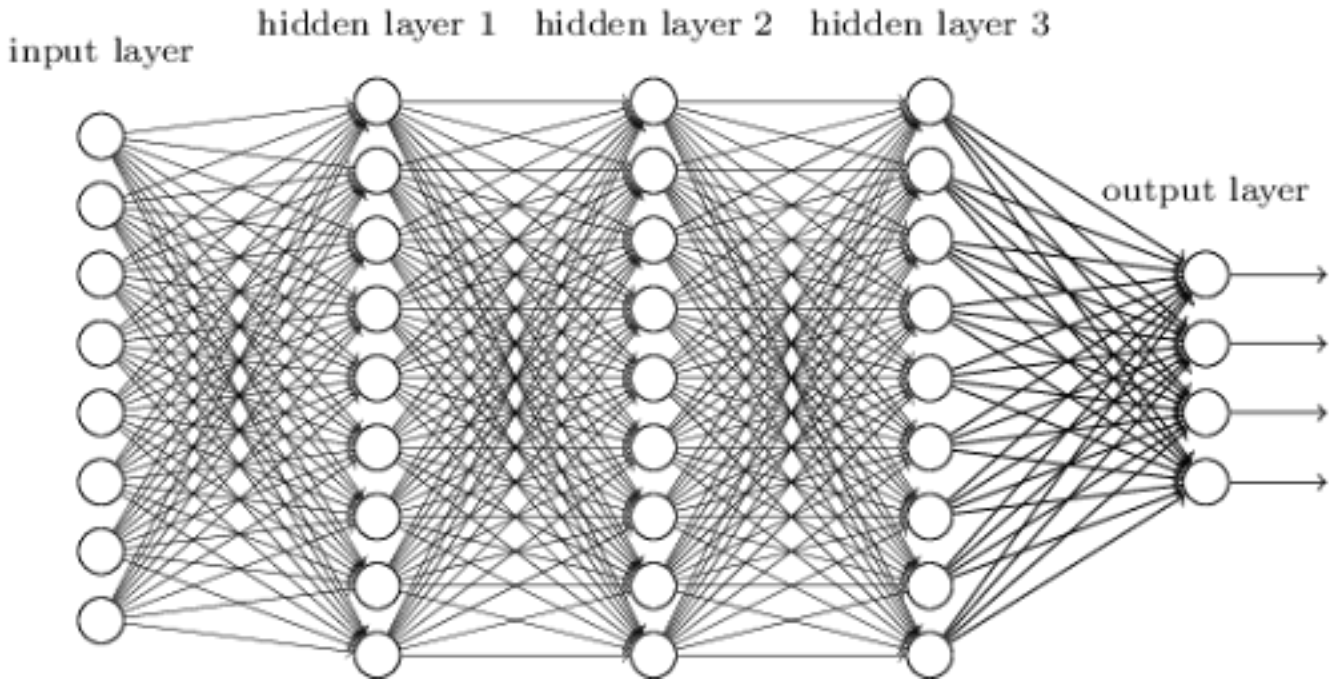
With the same number of total parameters, deep networks can learn more complex functions.

Recall that NNs are compositions of functions for which we are learning parameters:

$$f(g(h\left(i(j(x))\right)$$

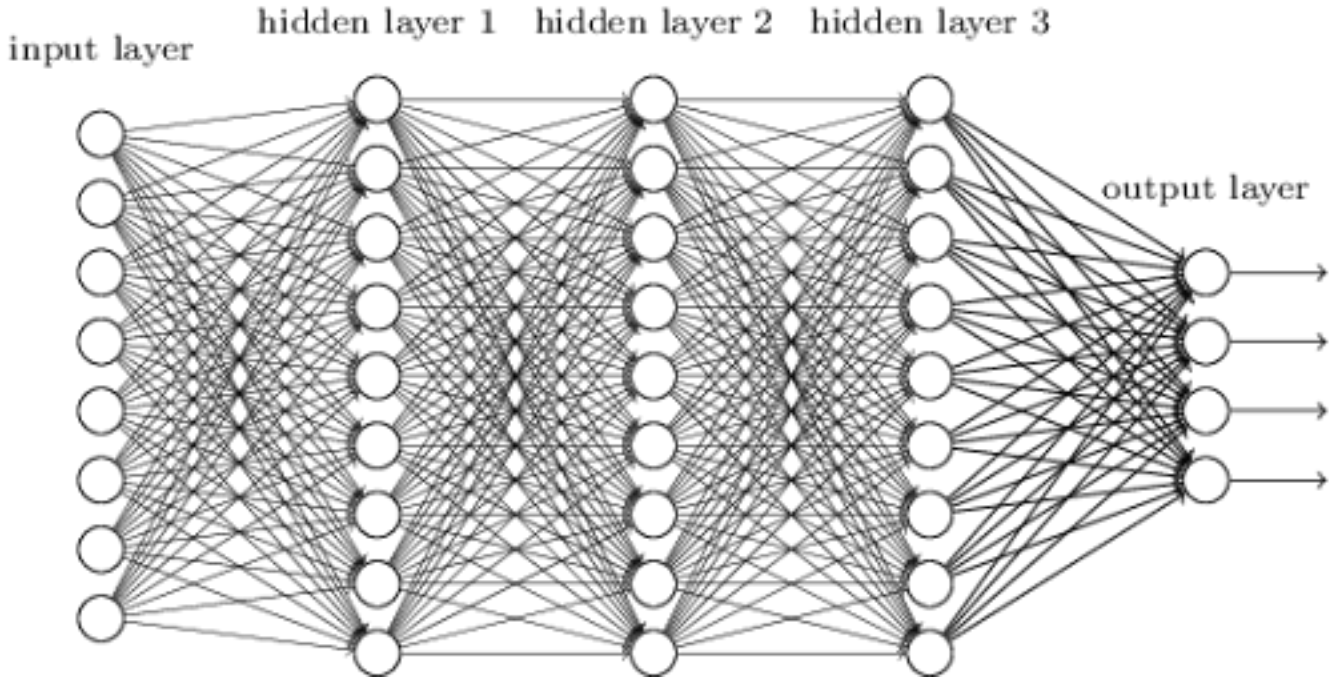
It's better (in general) to have more functions composed than it is to have more complex functions

- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?

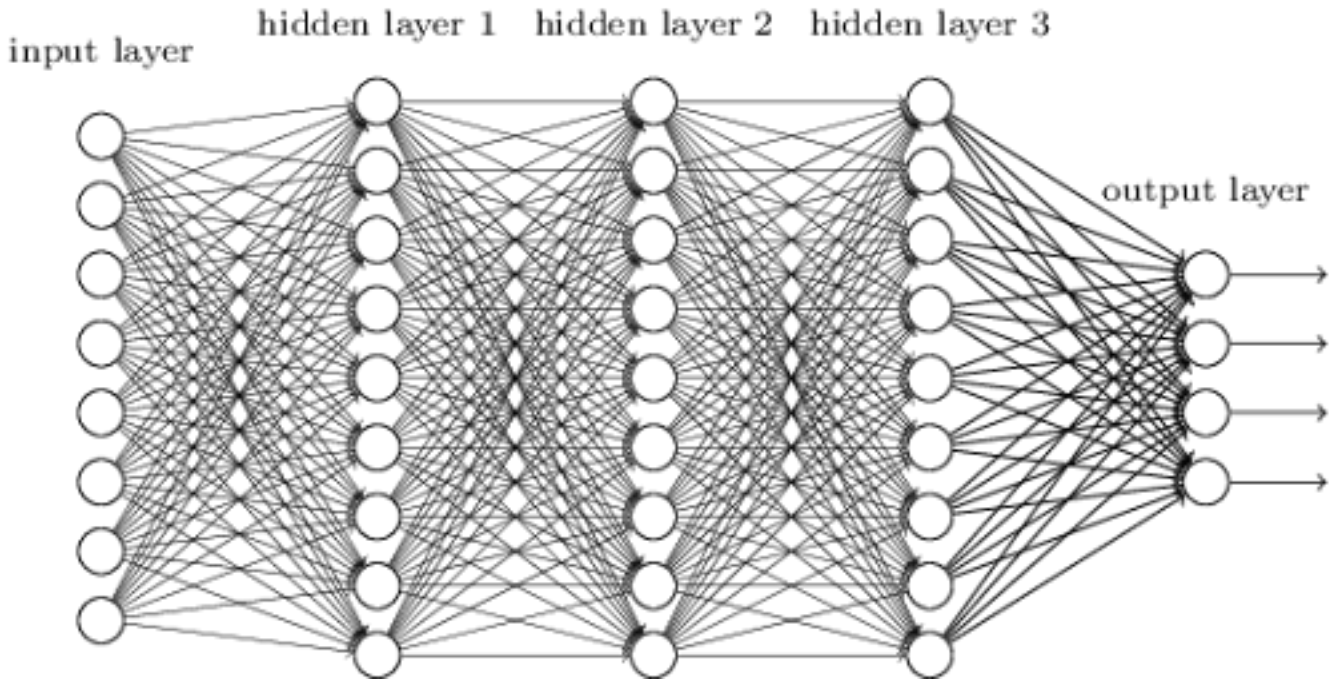


- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?

$W_1 \in \mathbb{R}^{10\times10}$
$W_2 \in \mathbb{R}^{10\times10}$
$W_3 \in \mathbb{R}^{10\times10}$
$W_4 \in \mathbb{R}^{10\times4}$
Total = 340

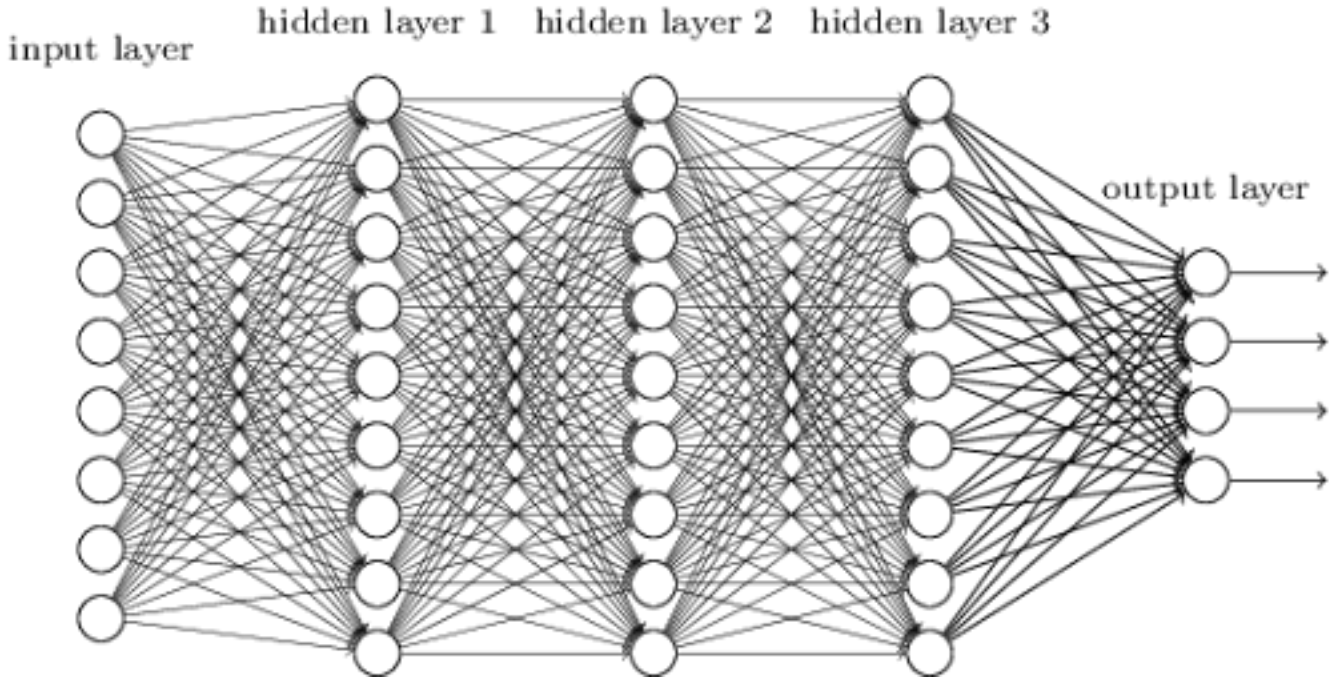


- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the width of each hidden layer?

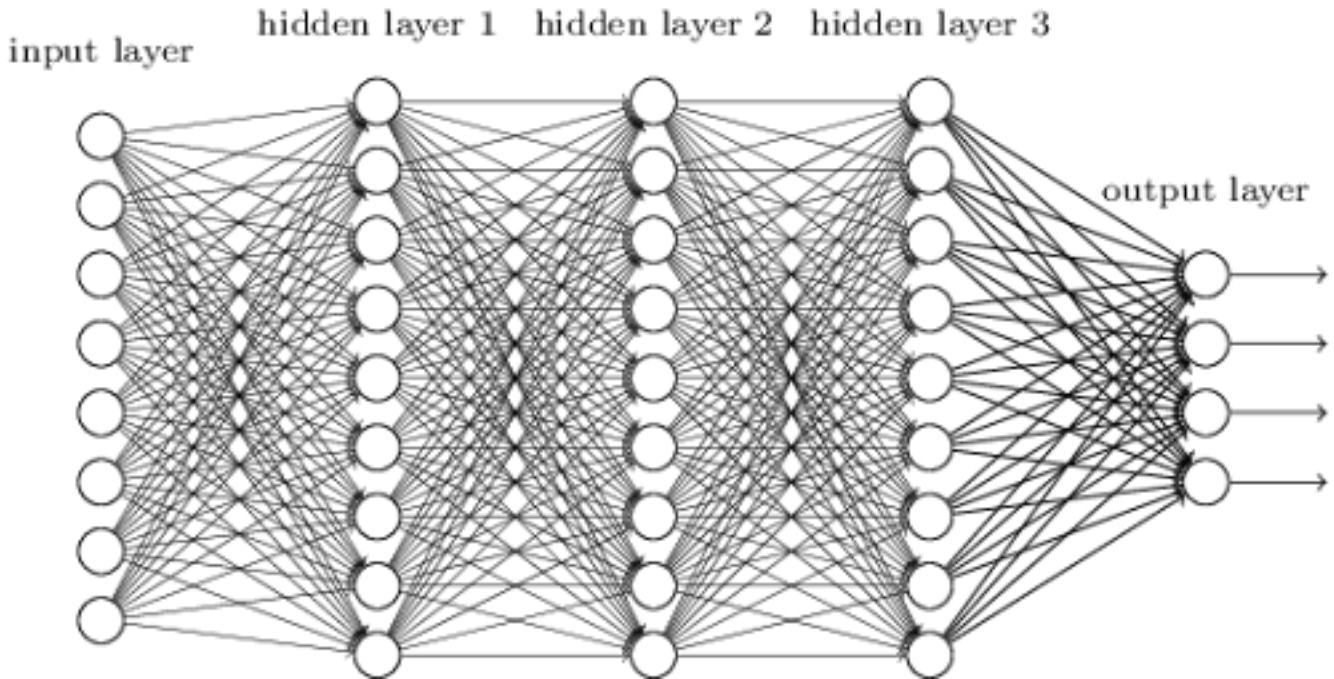


- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the width of each hidden layer?

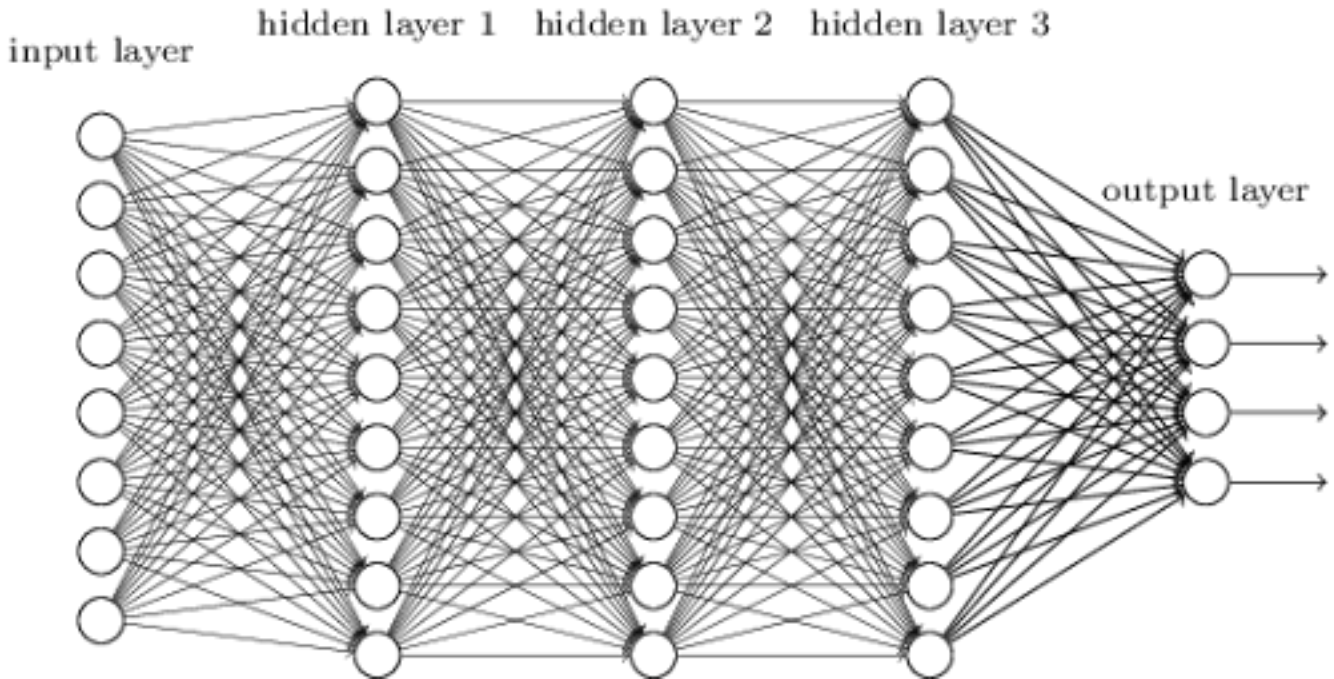
$W_1 \in \mathbb{R}^{10\times20}$
$W_2 \in \mathbb{R}^{20\times20}$
$W_3 \in \mathbb{R}^{20\times20}$
$W_4 \in \mathbb{R}^{20\times4}$
Total =1080



- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the depth? of each hidden layer?



- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the depth? of each hidden layer?



$W_1 \in \mathbb{R}^{10\times10}$
$W_2 \in \mathbb{R}^{10\times10}$
$W_3 \in \mathbb{R}^{10\times10}$
$W_4 \in \mathbb{R}^{10\times10}$
$W_5 \in \mathbb{R}^{10\times10}$
$W_6 \in \mathbb{R}^{10\times10}$
$W_7 \in \mathbb{R}^{10\times4}$
Total = 640

Overparameterization

Overparametization: Using more parameters than necessary for a ML problem.

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

~10,000 parameters in network

Overparameterization

Overparametization: Using more parameters than necessary for a ML problem.

Most of the time, networks use many more parameters than *necessary*.

In general, it's impossible to know the fewest amount of parameters that could solve a problem.

PLAYING ATARI WITH SIX NEURONS

Giuseppe Cuccu
eXascale Infolab
Department of Computer Science
University of Fribourg, Switzerland
name.surname@unifr.ch

Julian Togelius
Game Innovation Lab
Tandon School of Engineering
New York University, NY, USA
julian@togelius.com

Philippe Cudré-Mauroux
eXascale Infolab
Department of Computer Science
University of Fribourg, Switzerland
name.surname@unifr.ch

ABSTRACT

Deep reinforcement learning, applied to vision-based problems like Atari games, maps pixels directly to actions; internally, the deep neural network bears the responsibility of both extracting useful information and making decisions based on it. By separating the image processing from decision-making, one could better understand the complexity of each task, as well as potentially find smaller policy representations that are easier for humans to understand and may generalize better. To this end, we propose a new method for learning policies and compact state representations separately but simultaneously for policy approximation in reinforcement learning. State representations are generated by an encoder based on two novel algorithms: Increasing Dictionary Vector Quantization makes the encoder capable of growing its dictionary size over time, to address new observations as they appear in an open-ended online-learning context; Direct Residuals Sparse Coding encodes observations by disregarding reconstruction error minimization, and aiming instead for highest information inclusion. The encoder autonomously selects observations online to train on, in order to maximize code sparsity. As the dictionary size increases, the encoder produces increasingly larger inputs for the neural network: this is addressed by a variation of the Exponential Natural Evolution Strategies algorithm which adapts its probability distribution dimensionality along the run. We test our system on a selection of Atari games using tiny neural networks of only 6 to 18 neurons (depending on the game's controls). These are still capable of achieving results comparable—and occasionally superior—to state-of-the-art techniques which use two orders of magnitude more neurons.

Overparameterization

(This paper doesn't use SGD or backprop, but another optimization method)

PLAYING ATARI WITH SIX NEURONS

Giuseppe Cuccu
eXascale Infolab
Department of Computer Science
University of Fribourg, Switzerland
name.surname@unifr.ch

Julian Togelius
Game Innovation Lab
Tandon School of Engineering
New York University, NY, USA
julian@togelius.com

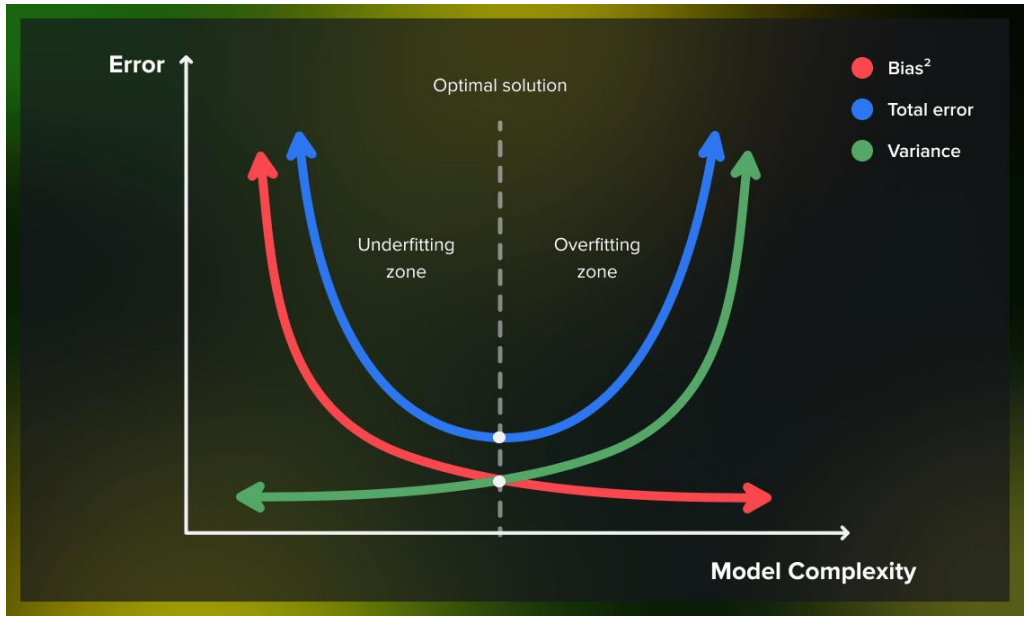
Philippe Cudré-Mauroux
eXascale Infolab
Department of Computer Science
University of Fribourg, Switzerland
name.surname@unifr.ch

ABSTRACT

Deep reinforcement learning, applied to vision-based problems like Atari games, maps pixels directly to actions; internally, the deep neural network bears the responsibility of both extracting useful information and making decisions based on it. By separating the image processing from decision-making, one could better understand the complexity of each task, as well as potentially find smaller policy representations that are easier for humans to understand and may generalize better. To this end, we propose a new method for learning policies and compact state representations separately but simultaneously for policy approximation in reinforcement learning. State representations are generated by an encoder based on two novel algorithms: Increasing Dictionary Vector Quantization makes the encoder capable of growing its dictionary size over time, to address new observations as they appear in an open-ended online-learning context; Direct Residuals Sparse Coding encodes observations by disregarding reconstruction error minimization, and aiming instead for highest information inclusion. The encoder autonomously selects observations online to train on, in order to maximize code sparsity. As the dictionary size increases, the encoder produces increasingly larger inputs for the neural network: this is addressed by a variation of the Exponential Natural Evolution Strategies algorithm which adapts its probability distribution dimensionality along the run. We test our system on a selection of Atari games using tiny neural networks of only 6 to 18 neurons (depending on the game's controls). These are still capable of achieving results comparable—and occasionally superior—to state-of-the-art techniques which use two orders of magnitude more neurons.

Overparameterization

Bias-Variance Tradeoff
(Traditional Understanding)



A Farewell to the Bias-Variance Tradeoff?
An Overview of the Theory of Overparameterized Machine Learning

Yehuda Dar* **Vidya Muthukumar†** **Richard G. Baraniuk‡**

Abstract

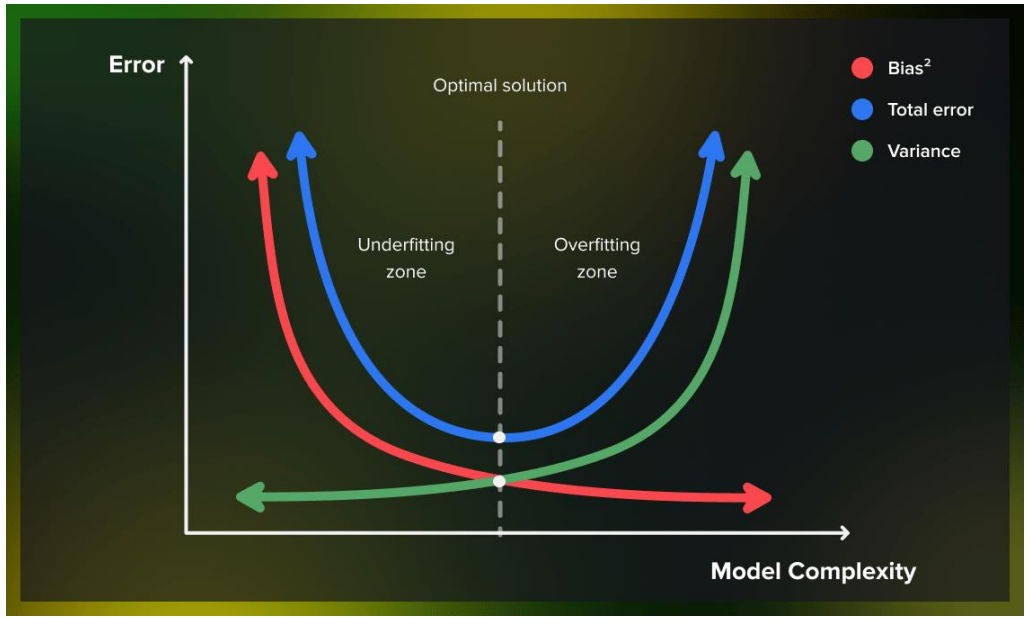
The rapid recent progress in machine learning (ML) has raised a number of scientific questions that challenge the longstanding dogma of the field. One of the most important riddles is the good empirical generalization of *overparameterized* models. Overparameterized models are excessively complex with respect to the size of the training dataset, which results in them perfectly fitting (i.e., *interpolating*) the training data, which is usually noisy. Such interpolation of noisy data is traditionally associated with detrimental overfitting, and yet a wide range of interpolating models – from simple linear models to deep neural networks – have recently been observed to generalize extremely well on fresh test data. Indeed, the recently discovered *double descent* phenomenon has revealed that highly overparameterized models often improve over the best underparameterized model in test performance.

Understanding learning in this overparameterized regime requires new theory and foundational empirical studies, even for the simplest case of the linear model. The underpinnings of this understanding have been laid in very recent analyses of overparameterized linear regression and related statistical learning tasks, which resulted in precise analytic characterizations of double descent. This paper provides a succinct overview of this emerging *theory of overparameterized ML* (henceforth abbreviated as TOPML) that explains these recent findings through a statistical signal processing perspective. We emphasize the unique aspects that define the TOPML research area as a subfield of modern ML theory and outline interesting open questions that remain.

https://serokell.io/blog/bias-variance-tradeoff

Overparameterization

Bias-Variance Tradeoff
(Traditional Understanding)



If you are overfitting, reduce model complexity (smaller width/fewer layers). If underfitting, add more model complexity.

https://serokell.io/blog/bias-variance-tradeoff

A Farewell to the Bias-Variance Tradeoff?
An Overview of the Theory of Overparameterized Machine Learning

Yehuda Dar* **Vidya Muthukumar†** **Richard G. Baraniuk‡**

Abstract

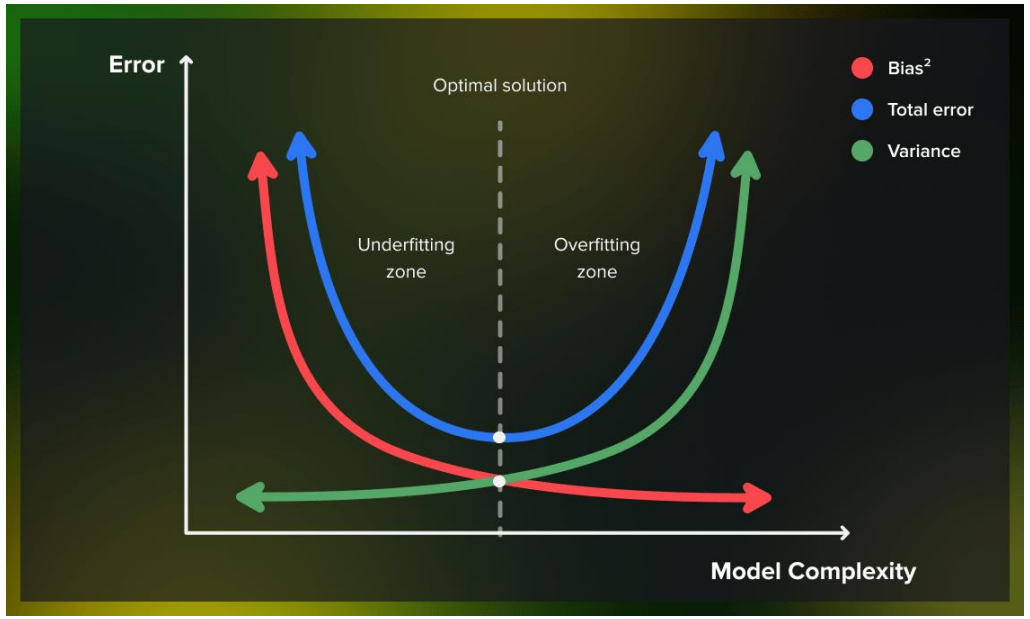
The rapid recent progress in machine learning (ML) has raised a number of scientific questions that challenge the longstanding dogma of the field. One of the most important riddles is the good empirical generalization of *overparameterized* models. Overparameterized models are excessively complex with respect to the size of the training dataset, which results in them perfectly fitting (i.e., *interpolating*) the training data, which is usually noisy. Such interpolation of noisy data is traditionally associated with detrimental overfitting, and yet a wide range of interpolating models – from simple linear models to deep neural networks – have recently been observed to generalize extremely well on fresh test data. Indeed, the recently discovered *double descent* phenomenon has revealed that highly overparameterized models often improve over the best underparameterized model in test performance.

Understanding learning in this overparameterized regime requires new theory and foundational empirical studies, even for the simplest case of the linear model. The underpinnings of this understanding have been laid in very recent analyses of overparameterized linear regression and related statistical learning tasks, which resulted in precise analytic characterizations of double descent. This paper provides a succinct overview of this emerging *theory of overparameterized ML* (henceforth abbreviated as TOPML) that explains these recent findings through a statistical signal processing perspective. We emphasize the unique aspects that define the TOPML research area as a subfield of modern ML theory and outline interesting open questions that remain.

(We will cover other techniques for managing overfitting next week)

Overparameterization

Bias-Variance Tradeoff
(Traditional Understanding)



If you are overfitting, reduce model complexity (smaller width/fewer layers). If underfitting, add more model complexity.

https://serokell.io/blog/bias-variance-tradeoff

A Farewell to the Bias-Variance Tradeoff?
An Overview of the Theory of Overparameterized Machine Learning

Yehuda Dar* **Vidya Muthukumar†** **Richard G. Baraniuk‡**

Abstract

The rapid recent progress in machine learning (ML) has raised a number of scientific questions that challenge the longstanding dogma of the field. One of the most important riddles is the good empirical generalization of *overparameterized* models. Overparameterized models are excessively complex with respect to the size of the training dataset, which results in them perfectly fitting (i.e., *interpolating*) the training data, which is usually noisy. Such interpolation of noisy data is traditionally associated with detrimental overfitting, and yet a wide range of interpolating models – from simple linear models to deep neural networks – have recently been observed to generalize extremely well on fresh test data. Indeed, the recently discovered *double descent* phenomenon has revealed that highly overparameterized models often improve over the best underparameterized model in test performance.

Understanding learning in this overparameterized regime requires new theory and foundational empirical studies, even for the simplest case of the linear model. The underpinnings of this understanding have been laid in very recent analyses of overparameterized linear regression and related statistical learning tasks, which resulted in precise analytic characterizations of double descent. This paper provides a succinct overview of this emerging *theory of overparameterized ML* (henceforth abbreviated as TOPML) that explains these recent findings through a statistical signal processing perspective. We emphasize the unique aspects that define the TOPML research area as a subfield of modern ML theory and outline interesting open questions that remain.

Optimizers

Optimizers

- SGD, SGD + Momentum, SGD + Adaptive Momentum (Adam), RMSProp,… the list is ever growing

Optimizers

- SGD, SGD + Momentum, SGD + Adaptive Momentum (Adam), RMSProp,… the list is ever growing

- How do you choose between them?

Optimizers

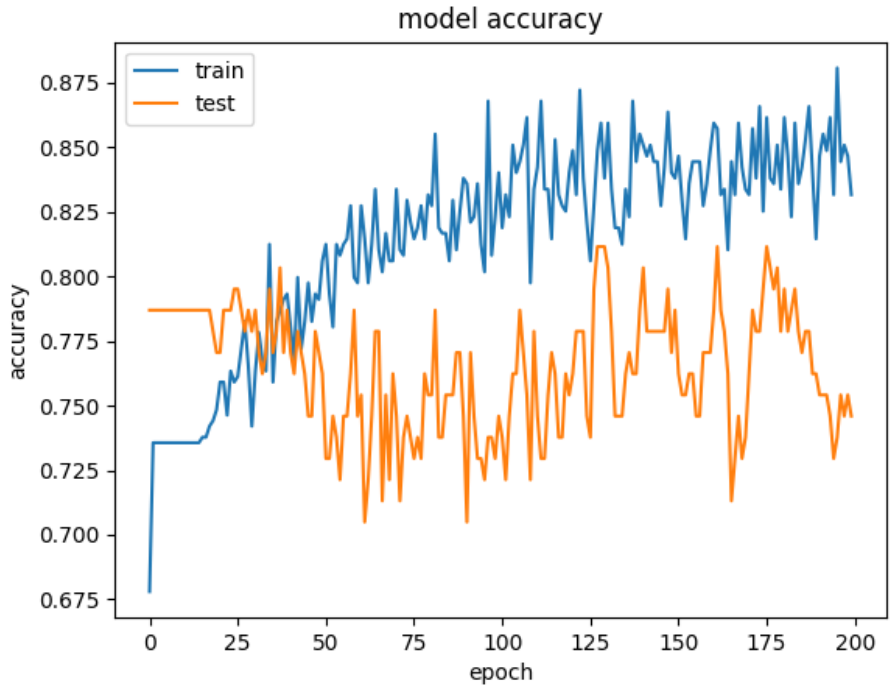
- SGD, SGD + Momentum, SGD + Adaptive Momentum (Adam), RMSProp,… the list is ever growing

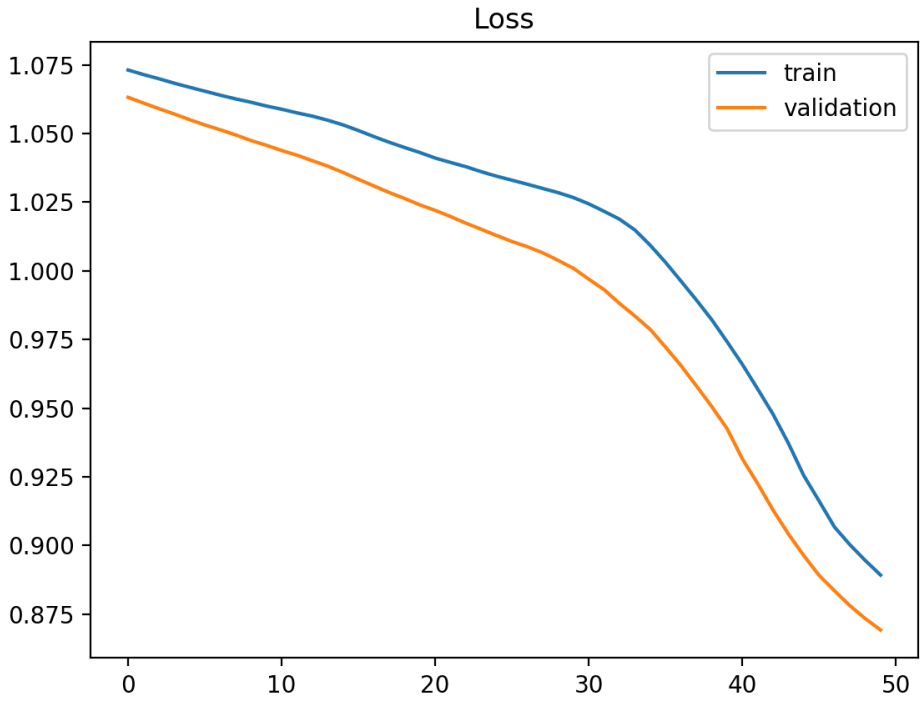
- How do you choose between them?

- Just use Adam.
  - The only downside is that it might work so well that you end up overfitting.
  - Suggested initial learning rate of 3e-4

Batch Size and Learning Rate

Having too small a batch or too high a learning rate can cause variance in training/validation loss – symptoms often look similar





General Tips

General Tips

- Don’t change too much at once.

General Tips

- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance

General Tips

- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance
- Don’t just randomly guess parameters, apply critical thinking, come up with a hypothesis and test your hypothesis.

General Tips

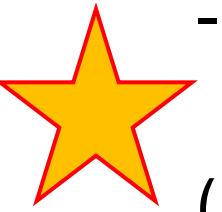
- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance
- Don’t just randomly guess parameters, apply critical thinking, come up with a hypothesis and test your hypothesis.

(Use the scientific method)

General Tips

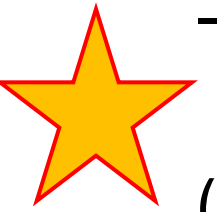
- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance
- Don’t just randomly guess parameters, apply critical thinking, come up with a hypothesis and test your hypothesis.

(Use the scientific method)



General Tips

- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance
- Don’t just randomly guess parameters, apply critical thinking, come up with a hypothesis and test your hypothesis.



(Use the scientific method)

Andrej Karpathy: A recipe for training neural networks
https://karpathy.github.io/2019/04/25/recipe/