Lecture 10: Neural Netwroks

Yen-Yi Ho

Department of Statistics

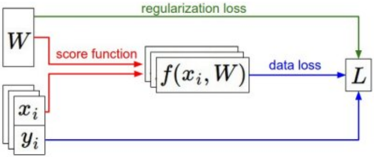
Recap

- We have some dataset of (X, Y)
- We have a score function
- We have a loss function

$$s_i = \mathbf{w}^\top \mathbf{x} + b$$

$$L_i = -log(\frac{e^{s_y}}{\sum_j e^{s_j}}) \quad \text{Softmax}$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W) \quad \text{Full loss}$$



The Loss Landscape: Optimize with Gradient Descent Variations

- Stochastic Gradient Descent (SGD)
- SGD + Momentum
- RMSProp
- Adam

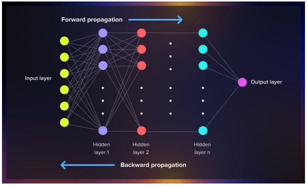
Outline for Today

- Overview
- Feedforward Neural Network Architecture
- Activation Functions
- Backpropagation

Training A Neural Network

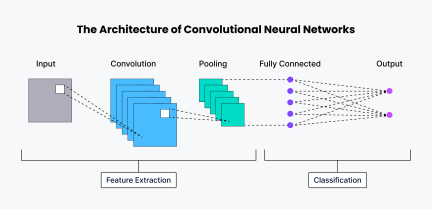
2 Steps:

- Forward propagation
  - The network makes predictions
  - Passing input data through the network layer by layer.
- Backpropagation
  - It is the process of adjusting weights of the network
  - Calculating the gradient of the loss w.r.t to weights
  - The weights are adjusted in the direction of reducing the loss



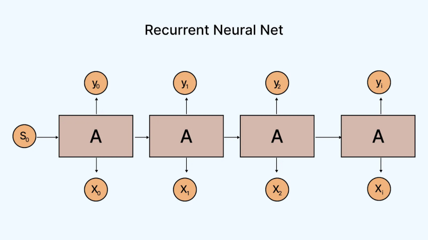
1 CNNs

- **Key Features**: Utilizes convolutional layers to process data ina grid pattern (like images).
- **Key Components**:
  - Convolutional Layers: Extract features from input images using filters.
  - Pooling Layers: Reduce dimensions and computational load, retaining key information.
  - Fully Connected Layers: Classify images based on extracted features.
- **Example Models**: CNNC, DeepBind

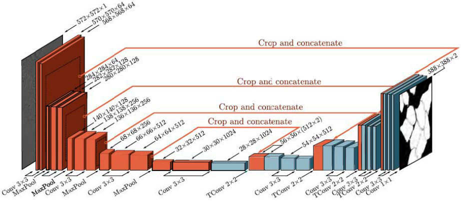


2 RNNs

- Key Features: Processes sequences of data (time-series data), with memory of previous inputs, capturing temporal dynamics.
- **Unique Feature:** Loop-like architecture allowing previous outputs to be used as inputs while having hidden states.
- Challenges & Solutions: Problem of vanishing gradients, memory consuming
- Example Models: LSTM (Long Short-Term Memory).

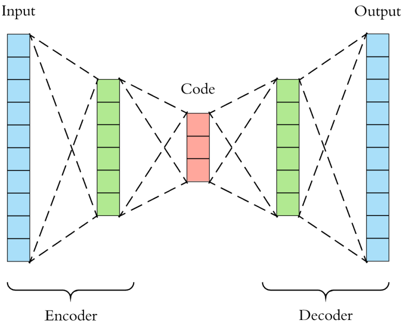


- **Key Features:** U-shaped architecture with symmetric encoder and decoder paths. Connections that concatenate feature maps from encoder to decoder
- **Structure:** Encoder: Series of convolutional and max-pooling layers that capture context. Bottleneck: Intermediate layer connecting encoder and decoder. Decoder: Series of up convolution and concatenation layers that restore resolution. Final Layer: Convolutional layer that maps features to the desired output.
- Example Models: Attention U-Net

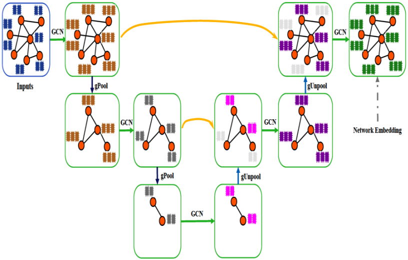


4 Autoencoders

- **Key Features:** Unsupervised dimensionality reduction and feature learning.
- **Structure:** Encoder compresses input; decoder reconstructs it.
- **Types:** Standard Autoencoders, Variational Autoencoders (VAEs).
- **Applications:** Denoising, anomaly detection in medical imaging.

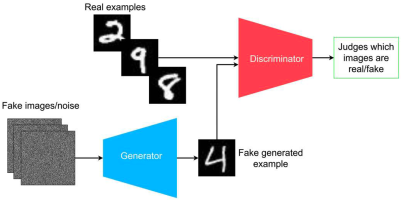


- **Key Features:** Operates on graph-structured data (nodes, edges).
- **Structure:** Nodes, Edges, Node Features, Graph Convolution, and Readout Layer.
- **Types:** GCN, GAT, GRN, Graph Autoencoders, Graph U-Net.
- **Applications:** Knowledge graphs, drug discovery, social networks.



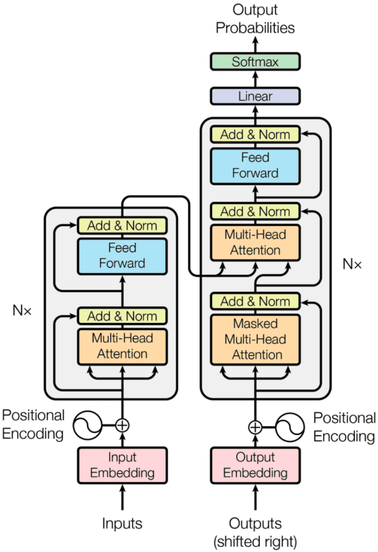
6 Generative Adversarial Networks (GANs)

- **Key Features:** Two networks (generator vs discriminator) in adversarial training.
- **Mechanism:** Generator tries to fool discriminator; discriminator learns to detect fakes.
- **Examples:** DCGAN, Pix2Pix, CycleGAN.
- **Applications:** Super-resolution, data augmentation in medical imaging.



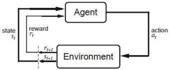
7 Transformer Models

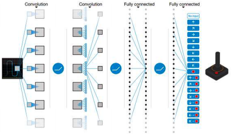
- **Key Features:** Self-attention mechanisms; handles sequences without recurrence.
- **Innovation:** Encoder-decoder structure removing recurrence/convolution.
- **Examples:** BERT (biomedical variants), AlphaFold.
- **Applications:** Genomic sequence analysis, protein structure prediction.

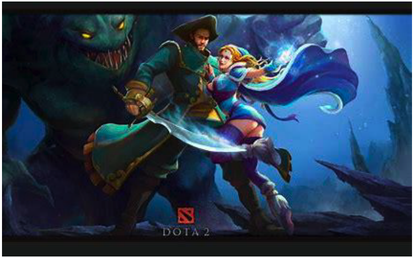


8 Deep Reinforcement Learning

- **Key Features:** Neural nets approximate value functions / policies from high-dimensional inputs.
- **Components:** Agent, Environment, Reward, Policy, Value function.
- **Examples:** DQN, A3C, PPO, SAC.
- **Applications:** Game playing, robotics, autonomous vehicles, healthcare.







Overview

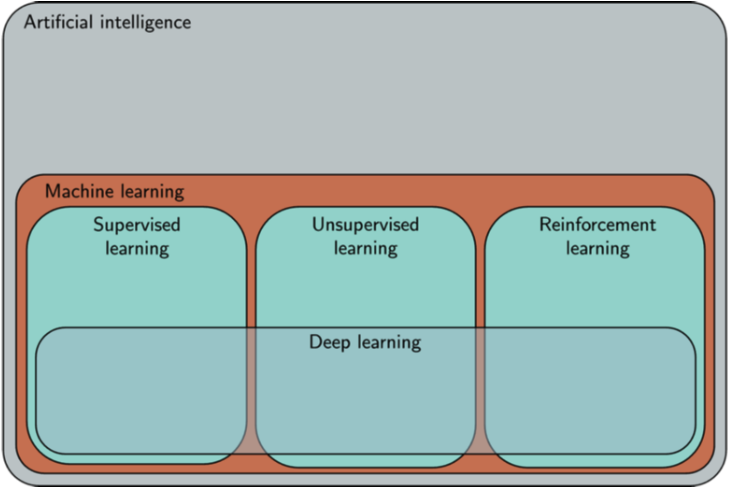
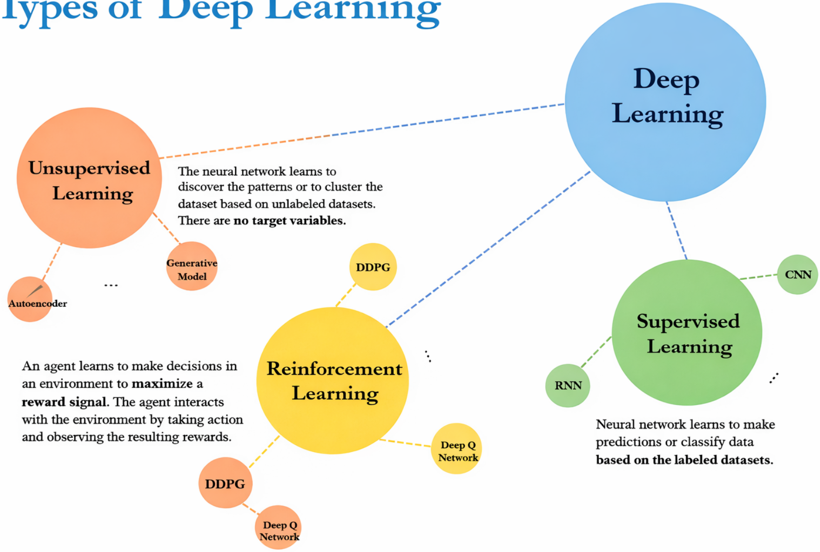


Figure adapted from Prince (2023)

Types of Deep Learning



Neural Networks

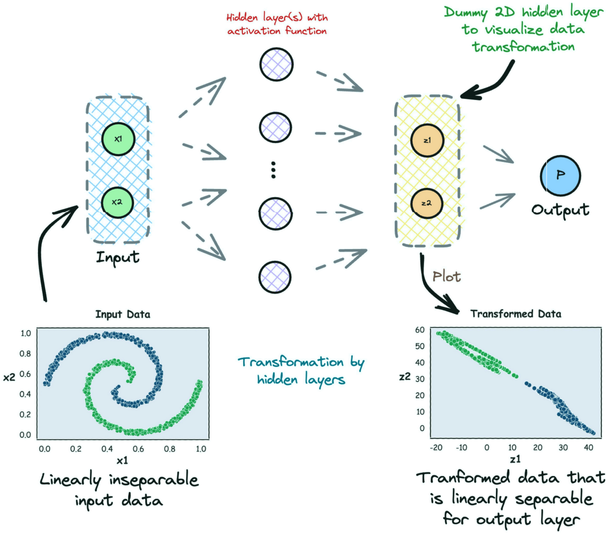
(Before) Linear Function: $f = Wx, x \in \mathbb{R}^G, W \in \mathbb{R}^{C \times G}$

(Now) 2-layer Neural Network: $f = W_2 max(0, W_1 x),$

$$x \in \mathbb{R}^G, W_1 \in \mathbb{R}^{H \times G}, W_2 \in \mathbb{R}^{C \times H}$$

(In practice, a learnable bias term is usually added to each layer as well)

Why do we want non-linearity



Neural Networks: Fully connected networks

(Before) Linear Function: $f = Wx, x \in \mathbb{R}^G, W \in \mathbb{R}^{C \times G}$

(Now) 2-layer Neural Network: $f = W_2 max(0, W_1 x),$

$$x \in \mathbb{R}^G, W_1 \in \mathbb{R}^{H \times G}, W_2 \in \mathbb{R}^{C \times H}$$

These are more accurately called "fully-connected networks" or "multi-layer perceptrons" (MLP)

Neural Networks: 3 layers

(Before) Linear Function: $f = Wx, x \in \mathbb{R}^G, W \in \mathbb{R}^{C \times G}$

(Now) 2-layer Neural Network: $f = W_2 max(0, W_1 x),$

$$x \in \mathbb{R}^G, W_1 \in \mathbb{R}^{H \times G}, W_2 \in \mathbb{R}^{C \times H}$$

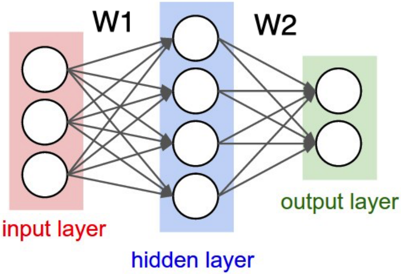
or 3-layer Neural Network: $f = W_3 max(0, W_2 max(0, W_1 x))$

$$x \in \mathbb{R}^G, W_1 \in \mathbb{R}^{H_1 \times G}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{C \times H_2}$$

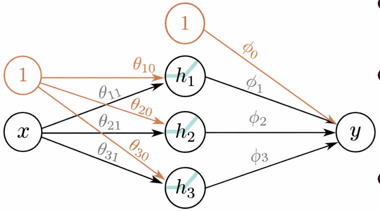
(Before) Linear Function: $f = Wx, x \in \mathbb{R}^G, W \in \mathbb{R}^{C \times G}$

(Now) 2-layer Neural Network: $f = W_2 max(0, W_1 x),$

$$x \in \mathbb{R}^G, W_1 \in \mathbb{R}^{H \times G}, W_2 \in \mathbb{R}^{C \times H}$$



Shallow Neural Network



$$h_d = a[\theta_{d0} + \sum_{i}^{D_i} \theta_{di} x_i]$$

$$y_i = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} h_d$$

Figure adapted from Prince (2023)

- **Neurons (Nodes)** receive input signals and perform and produce an output.
- **Channels (connections)** are associated with a weight value that determines the strength of the connection.
- **Bias** is conceptually similar to the intercept in linear regression , accounting for potential deviations from the ideal relationship between inputs and outputs.
- **Activation function** are threshold values that introduce non-linearities into the neural network, determining if the particular neuron will get activated or not.

Neural Network: Why is Max Operator Important?

(Before) Linear Function: $f = Wx, x \in \mathbb{R}^G, W \in \mathbb{R}^{C \times G}$

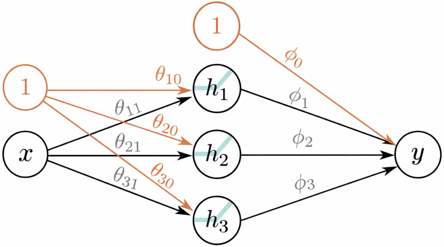
(Now) 2-layer Neural Network: $f = W_2 max(0, W_1 x),$

$$x \in \mathbb{R}^G, W_1 \in \mathbb{R}^{H \times G}, W_2 \in \mathbb{R}^{C \times H}$$

The function $max(0, z)$ is called the ReLU activation function.
Q: What if we try to build a neural network without one?

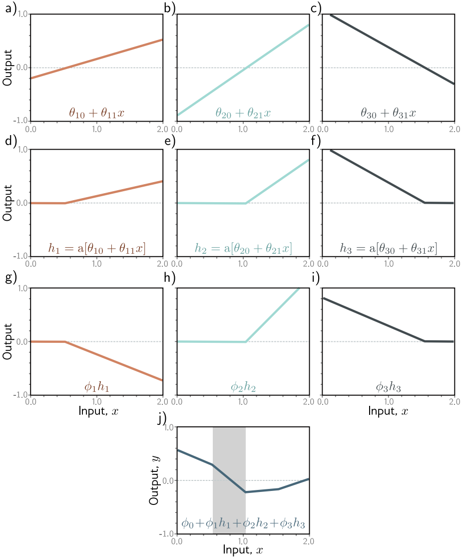
$$f = W_2 W_1 x$$

Example



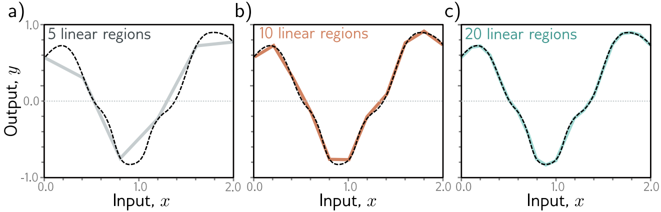
$$
\begin{aligned}
y &= \phi_0 + \phi_1 a[\theta_{10} + \theta_{11} x] + \phi_2 a[\theta_{20} + \theta_{21} x] + \phi_3 a[\theta_{30} + \theta_{31} x] \\
\phi &= (\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31})
\end{aligned}
$$

Example(Continued)



Universal approximation theorem

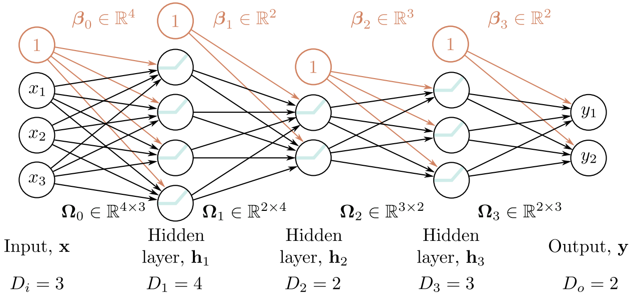
A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.



Hyperparameters

- The **width** (D) of a network refers to the number of hidden units in each layer.
- The **depth** (K) indicates the number of hidden layers.
- The overall **capacity** of a network is measured by its total number of hidden units.

Matrix notation for network with 3-hidden layers

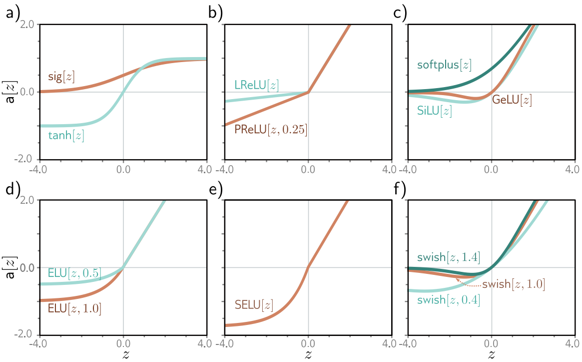


$$y = \beta_K + \Omega_K a[\beta_{K-1} + \Omega_{K-1} a[\ldots + \beta_2 + \Omega_2 a[\beta_1 + \Omega_1 a[\beta_0 + \Omega_0 x]]]]. \quad (1)$$

Outline for Today

- Overview
- Feedforward Neural Network Architecture
- **Activation Functions**
- Backpropagation

Activation Functions



- a) Logistic regression sigmoid and tanh functions
- b) Leaky ReLU and parametric ReLU with parameter 0.25.
- c) SoftPlus, Gaussian error linear unit, and sigmoid linear unit
- d) Exponential linear unit with parameters 0.5 and 1.0.
- e) Scaled exponential linear unit.
- f) Swish with parameters 0.4, 1.0, and 1.4.

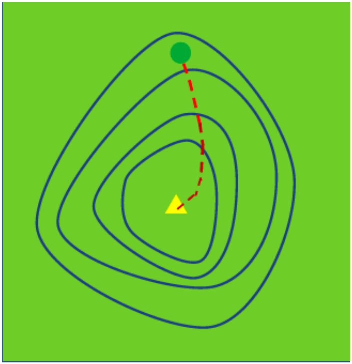
Activation Function Choice

- For binary classification: Use the **sigmoid** activation function in the output layer. It will squash outputs between 0 and 1, representing probabilities for the two classes.
- For multiclass classification: Use the **softmax activation** function in the output layer. It will output probability distributions over all classes.
- If unsure: Use the **ReLU activation** function in the hidden layers. ReLU is the most common default activation function and usually a good choice

Outline for Today

- Overview
- Feedforward Neural Network Architecture
- Activation Functions
- **Backpropagation**

Gradient Descent



$$\hat{W} = argmin \mathcal{L}[f(X;W), y]$$

$$W_{new} = W - \alpha \frac{\partial \mathcal{L}[f(X;W), y]}{\partial W}$$

Need

$$\frac{\partial \mathcal{L}[f(X;W), y]}{\partial W}$$

Counting Parameters in A Fully-Connected Network

We consider a **fully-connected (dense)** feedforward neural network with:

- **1 input node**
- **1 output node**
- K **hidden layers**
- Each hidden layer has D **nodes**, with $D > 2$
- **Bias parameters included** for every affine layer

Architecture (width D):

$$1 \rightarrow D \rightarrow D \rightarrow \cdots \rightarrow D \rightarrow 1$$

Parameter counting: layer by layer

For a dense layer mapping $n_{\text{in}} \to n_{\text{out}}$:

$$\#\text{weights} = n_{\text{in}} n_{\text{out}}, \quad \#\text{biases} = n_{\text{out}}$$

Parameter counting: layer by layer

For a dense layer mapping $n_{\text{in}} \to n_{\text{out}}$:

$$\#\text{weights} = n_{\text{in}} n_{\text{out}}, \quad \#\text{biases} = n_{\text{out}}$$

(1) Input $\to$ first hidden

$$(1 \to D): \quad \# = 1 \cdot D + D = 2D$$

Parameter counting: layer by layer

For a dense layer mapping $n_{\text{in}} \to n_{\text{out}}$:

$$\#\text{weights} = n_{\text{in}} n_{\text{out}}, \quad \#\text{biases} = n_{\text{out}}$$

(1) Input → first hidden

$$(1 \to D): \quad \# = 1 \cdot D + D = 2D$$

(2) Hidden → hidden

There are $(K-1)$ transitions of $(D \to D)$:

$$(D \to D): \quad \# = D^2 + D$$

$$\Rightarrow \quad (K-1)(D^2 + D)$$

Parameter counting: layer by layer

For a dense layer mapping $n_{\text{in}} \to n_{\text{out}}$:

$$\#\text{weights} = n_{\text{in}} n_{\text{out}}, \quad \#\text{biases} = n_{\text{out}}$$

(1) Input → first hidden

$$(1 \to D): \quad \# = 1 \cdot D + D = 2D$$

(2) Hidden → hidden

There are $(K-1)$ transitions of $(D \to D)$:

$$(D \to D): \quad \# = D^2 + D$$

$$\Rightarrow \quad (K-1)(D^2 + D)$$

(3) Last hidden → output

$$(D \to 1): \quad \# = D \cdot 1 + 1 = D + 1$$

Total parameter count

Add all contributions:

$$\text{Total} = 2D + (K-1)(D^2 + D) + (D+1)$$

Total parameter count

Add all contributions:

$$\text{Total} = 2D + (K-1)(D^2 + D) + (D+1)$$

Simplify:

$$\boxed{\text{Total} = (K-1)D^2 + (K+2)D + 1}$$

- Dominant term: $\boxed{(K-1)D^2}$ so scaling is $\boxed{O(KD^2)}$

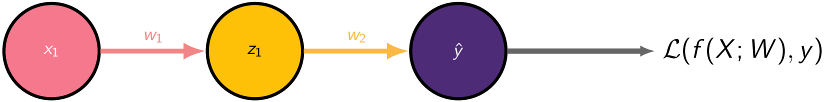
Worked example

Let $K = 3$ hidden layers and $D = 10$ nodes per hidden layer.

$$\text{Total} = (3-1)\cdot 10^2 + (3+2)\cdot 10 + 1 = 2\cdot 100 + 50 + 1 = \boxed{251}$$

Connection	Weights	Biases
$1 \to D$	10	10
$D \to D$ (2 transitions)	$2 \cdot 100$	$2 \cdot 10$
$D \to 1$	10	1
Total	220	31

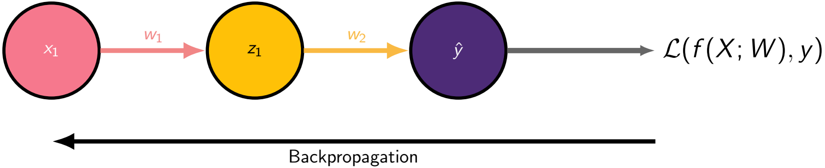
Backpropagation



$$\frac{\partial \mathcal{L}[f(X;W),y]}{\partial w_1} = \frac{\partial \mathcal{L}[f(X;W),y]}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_1} \qquad \text{chain rule}$$

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

Backpropagation



$$\frac{\partial \mathcal{L}[f(X;W),y]}{\partial w_1} = \frac{\partial \mathcal{L}[f(X;W),y]}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_1} \qquad \text{chain rule}$$

$$\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$