Lecture 3

MLP

What is an MLP (and why it works)

- **MLP = feed-forward network**: layers of neurons, each layer fully connected to the previous one.
- **Key ingredients**: linear transforms + **nonlinear** activations. Without nonlinearity, a stack of layers collapses to one linear map.
- **Power**: With enough hidden units, an MLP can approximate any continuous function on a compact set (Universal Approximation Theorem).

Anatomy of a layer

For layer l:

$$\mathbf{z}^{[l]} = W^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}, \quad \mathbf{a}^{[l]} = f^{[l]}(\mathbf{z}^{[l]})$$

- $W^{[l]} \in \mathbb{R}^{m_l \times m_{l-1}}$, $\mathbf{b}^{[l]} \in \mathbb{R}^{m_l}$.
- $\mathbf{a}^{[0]} = \mathbf{x}$ (input), $\mathbf{a}^{[L]}$ is output.

Activations (and when to use them)

- **ReLU**: $f(z) = \max(0, z)$. Fast, sparse gradients; watch "dead" neurons (use good init).
- **Leaky/Parametric ReLU**: fixes dead ReLU ($f(z) = \max(\alpha z, z)$).
- **Tanh**: zero-centered; can saturate (small gradients at large $|z|$).
- **Sigmoid**: for probabilities in **binary** output; avoid in deep hidden layers (saturation).
- **Softmax**: multi-class probabilities: $\hat{y}_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$.

Losses (likelihood view)

- **Binary classification** (target $y \in \{0, 1\}$, output $\hat{y} = \sigma(z)$):

$$\mathcal{L}(\hat{y}, y) = -\big[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\big]$$

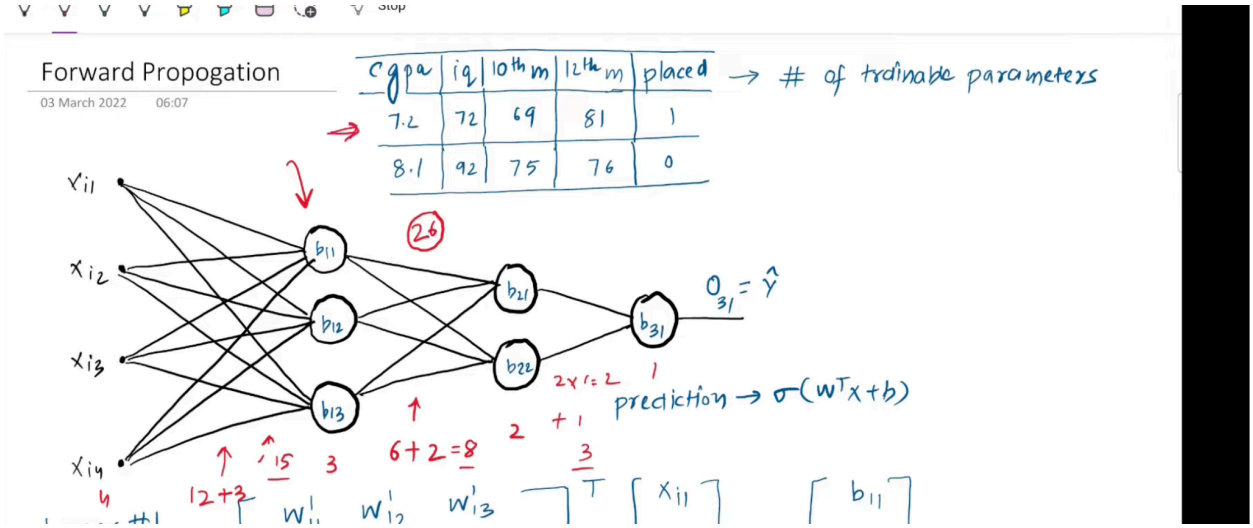
- **Multi-class** (one-hot y, softmax $\hat{\mathbf{y}}$):

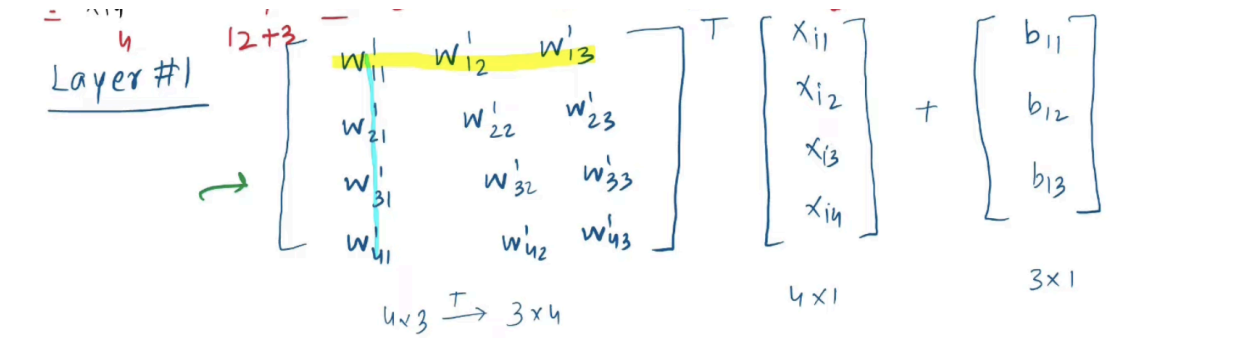
$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_k y_k \log \hat{y}_k$$

- **Regression**: $\frac{1}{2}||\hat{\mathbf{y}} - \mathbf{y}||^2$ or MAE/Huber.

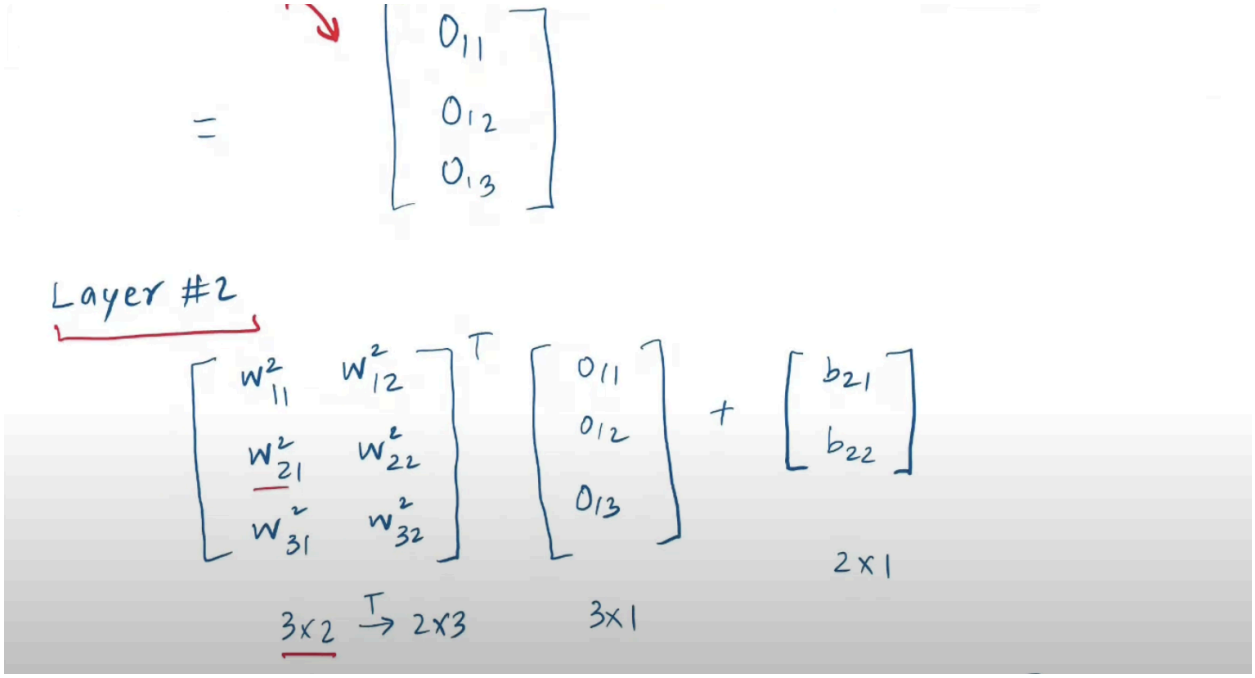
[A Neural Network Playground]

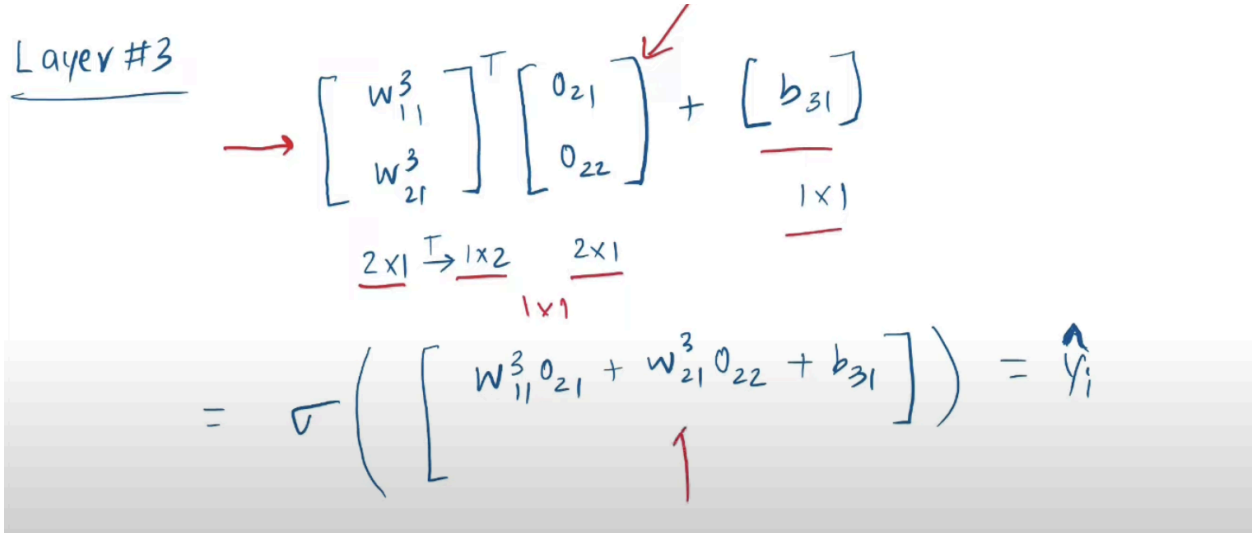
Forward Propagation





$$= \begin{bmatrix} w'_{11}x_{i1} + w'_{21}x_{i2} + w'_{31}x_{i3} + w'_{41}x_{i4} \\ w'_{12}x_{i1} + w'_{22}x_{i2} + w'_{32}x_{i3} + w'_{42}x_{i4} \\ w'_{13}x_{i1} + w'_{23}x_{i2} + w'_{33}x_{i3} + w'_{43}x_{i4} \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \end{bmatrix}$$



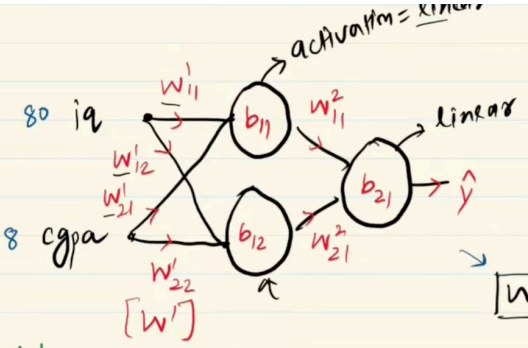


Backpropagation

Backprop

4 students

iq	cgpa	lpa
80	8	3
60	9	5
70	5	8
120	7	11



(W, b) Random val

$W \to 1 \;,\; b \to 0$

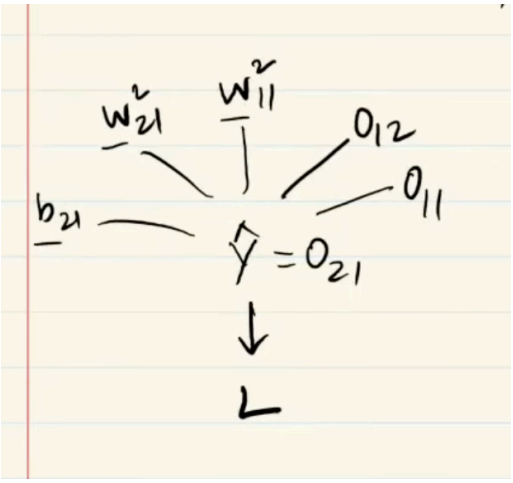
Steps → 0) init W, b

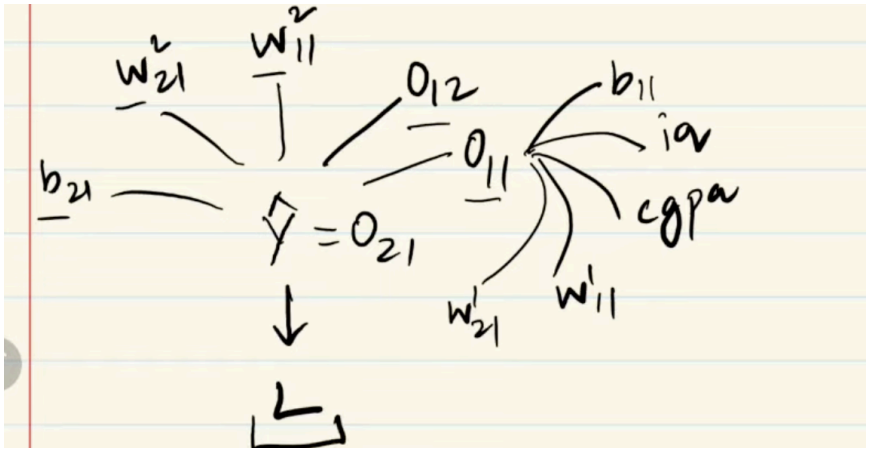
1) You select a point (row)
↳ student

2) Predict (lpa) → forward prop [Dot product]

3) Choose a loss function

$$O_{21} = W^2_{11} O_{11} + W^2_{21} O_{12} + b_{21}$$





4) Weights and bias update

↳ Gradient descent

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

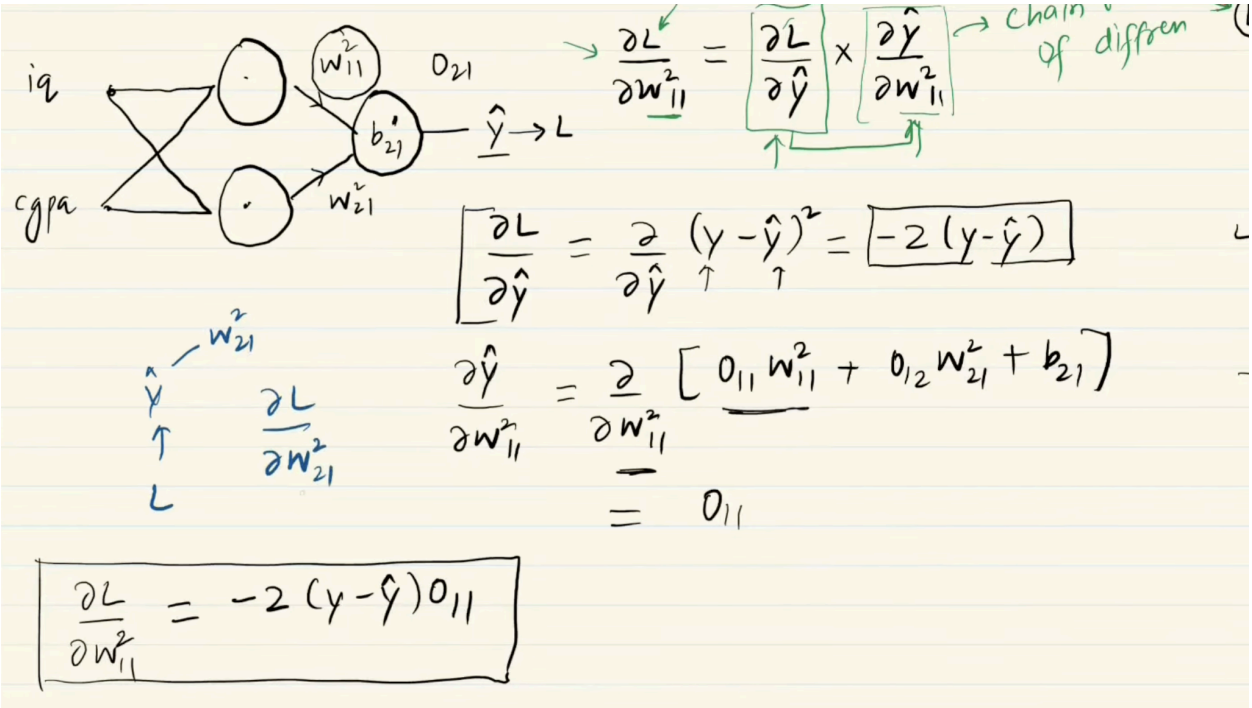
$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$

$$w^2_{11\,new} = w^2_{11\,old} - \eta \frac{\partial L}{\partial w^2_{11}}$$

$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$

$$w^2_{21\,new} = w^2_{21\,old} - \eta \frac{\partial L}{\partial w^2_{21}}$$

$$b_{21\,new} = b_{21\,old} - \eta \frac{\partial L}{\partial b_{21}}$$



Note

The **Universal Approximation Theorem (UAT)** is one of the most important results in neural network theory. It formally states that **a feedforward neural network with at least one hidden layer, containing a finite number of neurons, can approximate any continuous function to any desired degree of accuracy**—provided the activation function satisfies certain conditions (nonlinear, bounded, and continuous, like sigmoid, tanh, or even ReLU in some formulations).

Intuition

- Imagine you want a neural network to "learn" any curve or surface.
- The theorem says: **as long as you allow enough hidden neurons, the network can approximate the function arbitrarily well.**
- It doesn’t say how many neurons are needed, nor how to train the network efficiently—it just guarantees that such a network exists.