# Tutorial 2A

| | |
|---|---|
| course | 🧠 Deep Learning |
| Date | @February 9, 2026 |
| Status | Done |

# Questions

## Question 1

Optimization dynamics as inductive bias (*Goodfellow et al., Chapter 8.1–8.3*). Consider a neural network with parameters $\theta$ trained using gradient-based methods.

**a) Write down the parameter update rules for:**

• Stochastic Gradient Descent (SGD)

setup

parameters: $\theta$

learning rate : $\epsilon_i$

iteration : $k = 1$

while stoping criteria are not met:

sample mini. batch of $m$ examples from the training set $\{x^{(1)}, x^{(2)}, \ldots, x^{m}\}$

with corresponding $y^{(i)}$

compute gradient estimate : $\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L(f(x_{(i)}; \theta), y^{(i)})$

apply update: $\theta = \theta - \epsilon_k \hat{g}$

k +=1

end while

• SGD with Momentum,

setup: learniragte $\epsilon$ and moementum parameter $\alpha$.

initial parameter $\theta$, initial velocity $v$.

while creterion not met do:

sample mini. batch of $m$ examples from the training set $\{x^{(1)}, x^{(2)}, \ldots, x^{m}\}$

with coresponding $y^{(i)}$

compute gradient estimate: $g = \frac{1}{m} \nabla_\theta \sum_i L(f(x_{(i)}; \theta), y^{(i)})$

compute velocity: $v = \alpha v - \epsilon g$.

apply update = $\theta = \theta + v$

• Adam aka adaptaitve moiementum

step size $\epsilon$

exponetial decay rates for moment estimate, $p_1$ and $p_2 \in [0, 1)$

small constant : $\delta$ (e-8)

initial parameter : $\theta$

inisitlise 1sr and seconde moment variable $s = 0, r = 0$

while criterion not met do:

sample mini. batch of $m$ examples from the training set $\{x^{(1)}, x^{(2)}, \ldots, x^{m}\}$ and corresponding $y^{(i)}$

compute gradient estimate: $g = \frac{1}{m} \nabla_\theta \sum_i L(f(x_{(i)}; \theta), y^{(i)})$

update biased first moment estimate: $s = p_1 s + (1 - p_1) g$

update biased second moment estimate: $r = p_2 r + (1 - p_2) g \odot g$

correct bias in first moment: $\hat{s} = \frac{s}{1 - p_1^t}$

correct bias in second moment: $\hat{r} = \frac{r}{1 - p_2^t}$

compute update: $\delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$

apply update: $\theta = \theta + \Delta\theta$

**B) or each method, identify:**

- what information from past gradients is retained,
- how gradient noise is treated,
- how step sizes are adapted over time.

**SDG**:

Past gradients retained: none, only the current mini batch gradient.
Gradient noise: not explicitly treated, you just live with the mini batch noise.
Step sizes over time: not adapted automatically, you choose a schedule for \epsilon_k or keep it constant.

**SDG + momentum:**

Past gradients retained: an exponential moving average of past gradients via v.
Gradient noise: smoothed by the moving average, reduces zig zag and helps consistent directions accumulate.
Step sizes over time: still not automatically adapted per parameter, \epsilon is global and fixed or scheduled by you. Momentum changes the effective step through v, but it is not a true adaptive learning rate.

**adam:**

Past gradients retained: two exponential moving averages, one of gradients s and one of squared gradients r.
Gradient noise: smoothed by s, and noisy dimensions get down weighted because large variance increases r.
Step sizes over time: adapted per parameter as $\epsilon / (\sqrt{\hat{r}_k} + \delta)$; also bias correction makes early steps not artificially too small

**C)Consider a fixed loss function $L(\theta)$ used to train a neural network. Explain what information the loss function provides to the learning algorithm, and what it does not specify about the internal structure of the network**

the loss function gices the learning algorythm a scalar mesure of how good the curent parameters are with respect to the objecive. it give no information on the network acritecture or how to optimise the parameters.

**D) Discuss how each update rule introduces an inductive bias on the learning dynamics. In particular, comment on:**
**• sensitivity to curvature of the loss landscape,**
**• robustness to noisy gradients,**
**• potential effects on the representations learned.**

## SGD

**Sensitivity to curvature of the loss landscape**

SGD uses the same learning rate in all directions, so it is highly sensitive to curvature. In steep directions it can oscillate or diverge, while in flat directions progress is slow. It does not account for anisotropy of the landscape.

**Robustness to noisy gradients**

Gradient noise is not explicitly handled. The stochasticity from mini batches introduces noise that can help escape sharp minima, but updates are directly affected by variance in the gradient.

**Potential effects on learned representations**

SGD tends to favor flatter minima because noisy updates make sharp minima unstable. This implicit bias is often associated with better generalization and simpler, smoother representations.

## SGD with Momentum

**Sensitivity to curvature of the loss landscape**

Momentum reduces sensitivity to high curvature directions by accumulating gradients over time. It damps oscillations in steep directions and accelerates progress along low curvature valleys.

**Robustness to noisy gradients**

More robust than plain SGD. The exponential averaging of gradients filters out high frequency noise and reinforces consistent directions.

**Potential effects on learned representations**

Momentum biases learning toward representations that align with persistent gradient directions. This can lead to faster formation of higher level features and more stable internal representations compared to SGD.

## Adam

**Sensitivity to curvature of the loss landscape**

Adam adapts the step size per parameter using second moment estimates, making it much less sensitive to curvature. Parameters in steep directions automatically get smaller steps.

**Robustness to noisy gradients**

Highly robust. The moving average of squared gradients downweights parameters with high variance, effectively suppressing noise dominated directions.

**Potential effects on learned representations**

Adam tends to learn representations faster and more uniformly across parameters. However, it can converge to sharper minima and different internal representations than SGD, sometimes with worse generalization due to reduced pressure toward flat solutions.

### Big picture takeaway

The update rule is not neutral.

SGD implicitly favors flat minima and simpler representations.

Momentum adds directional persistence and stability.

Adam strongly biases optimization toward fast, curvature aware convergence, often at the cost of weaker implicit regularization.

This is exactly why **optimization dynamics themselves act as an inductive bias**, even with the same loss and architecture.

**E) The same loss function can be minimized using different optimization algorithms. Explain why using SGD, Momentum, or Adam can lead to different parameter trajectories and different internal representations, even when optimizing the same Loss.**

Even with the same loss, different optimizers follow different parameter trajectories because they transform and accumulate gradients differently and interact with the non convex loss landscape in distinct ways.
This leads them to converge to different minima, since optimization history, noise handling, and step scaling differ.
As a result, the network can learn different internal representations and generalize differently despite achieving similar loss values.

## Question 2

Initialization and signal propagation in deep networks (Goodfellow et al., Chapter 8.3,
batch normalization is introduced in 8.7.).
Consider a deep feedforward neural network composed of linear layers followed by
nonlinear activation functions.
**(a) Explain qualitatively why naive random initialization can lead to vanishing or exploding activations and gradients as depth increases.**

With naive initialization, each layer multiplies activations by random weights.
Across many layers, these repeated multiplications make activations and gradients either shrink toward zero (vanishing) or grow uncontrollably (exploding), making training unstable or impossible.

(b) Assume weights are initialized independently with zero mean and fixed variance.
Describe how the variance of activations propagates forward through layers, and
how the variance of gradients propagates backward.

- **Forward pass**: activation variance at layer $\ell$ depends on weight variance and activation function. If variance > 1, activations explode; if < 1, they vanish.

- **Backward pass**: gradient variance propagates similarly through transposed weights and activation derivatives, leading to vanishing or exploding gradients.

(c) Explain the motivation behind Xavier and He initialization schemes. What quantities are they designed to preserve, and why does the choice of activation function matter?

These schemes choose weight variance to **preserve variance across layers**.

- **Xavier** balances variance for linear or tanh activations.
- **He** preserves variance for ReLU, which discards half the signal.

  Activation choice matters because it changes how variance flows through the network.

(d) Consider a deep network that is expressive enough to represent the target function. Explain why, despite this expressive power, poor initialization can prevent the network from learning useful internal representations.

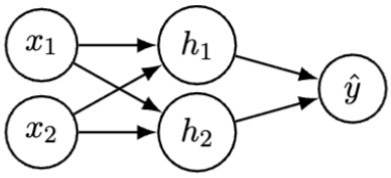
Even if the network can represent the target function, poor initialization can cause gradients to vanish or explode.
As a result, parameters cannot move into useful regions of the loss landscape, so meaningful representations never form.

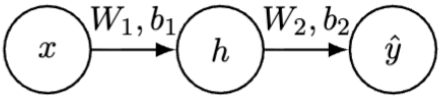
(e) Batch normalization is often introduced as a mechanism to stabilize training in deep networks. Explain how normalizing layer activations during training affects forward signal propagation and backward gradient flow, and why this can reduce sensitivity to initialization.

Batch normalization normalizes activations to have stable mean and variance during training.
This stabilizes forward signals and backward gradients, allowing deeper networks to train reliably and reducing dependence on careful initialization.

## Question 3

1. Explain what each node represents in the unit-level graph and in the layer-level graph.




### Inputs

- $x = (x_1, x_2) \in \mathbb{R}^2$

### Hidden layer

- Hidden units:

  $h = (h_1, h_2) \in \mathbb{R}^2$
- Weights from input to hidden:

  $W_1 = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} \in \mathbb{R}^{2\times 2}$
- Bias:

  $b_1 = (b_{1,1}, b_{1,2}) \in \mathbb{R}^2$
- Pre-activation:

  $z = W_1 x + b_1$
- Activation:

  $h = \phi(z)$

  where $\phi(\cdot)$ is ReLU or tanh (applied element-wise).

### Output layer

- Weights:

  $W_2 = (w_{2,1}, w_{2,2}) \in \mathbb{R}^{1\times 2}$
- Bias:

  $b_2 \in \mathbb{R}$
- Pre-activation:

  $a = W_2 h + b_2$
- Output:

  $\hat{y} = \sigma(a)$

  where $\sigma(\cdot)$ is the sigmoid.

### Compact layer graph

This is the **same network**, but expressed at the **layer level**.

### Input

- $x \in \mathbb{R}^2$

### Hidden layer

- Parameters: $W_1 \in \mathbb{R}^{2\times 2},\ b_1 \in \mathbb{R}^2$
- Computation:

  $h = \phi(W_1 x + b_1)$

### Output layer

- Parameters: $W_2 \in \mathbb{R}^{1\times 2},\ b_2 \in \mathbb{R}$

- Computation:

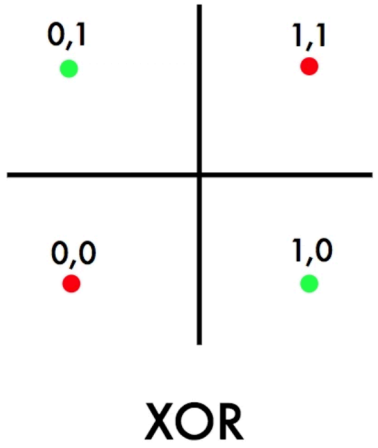
  $\hat{y} = \sigma(W_2 h + b_2)$

3. Layer-level drawings scale and support theoretical reasoning in deep networks, while unit-level drawings expose individual computations and are useful for small models or debugging.

(b) Linear separability and representation change.

Show that the XOR problem is not linearly separable in the input space.

can prove this graphically




XOR

and algebraically:

$(0,0) \mapsto 0, \quad (1,0) \mapsto 1, \quad (0,1) \mapsto 1, \quad (1,1) \mapsto 0.$

Assume, for contradiction, that XOR is linearly separable. Then there exists a linear classifier

$\hat{y} = \mathbf{1}[w^\top x + b > 0], \quad w = (w_1, w_2).$

To classify correctly, we need

$w^\top(0,0) + b = b < 0$
$w^\top(1,0) + b = w_1 + b > 0$
$w^\top(0,1) + b = w_2 + b > 0$
$w^\top(1,1) + b = w_1 + w_2 + b < 0.$

Now add the two positive inequalities:

$(w_1 + b) + (w_2 + b) > 0 \;\Rightarrow\; w_1 + w_2 + 2b > 0$
$\Rightarrow\; w_1 + w_2 + b > -b.$

But from $b < 0$, we have $-b > 0$, so this implies

$w_1 + w_2 + b > 0,$

which contradicts the required condition $w_1 + w_2 + b < 0.$

**C) the loss function used for XOR classification (e.g. cross-entropy) does not explicitly encode the notion of hidden representations. Explain how minimizing this loss nevertheless leads to the emergence of an internal representation that makes the problem linearly separable.**

Although the loss is defined only on the output, backpropagation forces hidden layers to learn nonlinear transformations that reshape the input space so that the classes become linearly separable at the output.

- The loss defines *what* is correct.
- The architecture and optimization determine *how* correctness is achieved.
- Internal representations emerge as **solutions to the optimization problem**, not as explicit objectives.

d) Optimization and gradient flow.

- Explain why unstable gradient flow can prevent the hidden layer from learning a useful representation.

Unstable gradient flow refers to **vanishing or exploding gradients** during backpropagation:

- **Vanishing gradients**: gradients shrink toward zero.
- **Exploding gradients**: gradients grow uncontrollably.

Unstable gradient flow prevents the hidden layer from receiving meaningful learning signals, so its parameters cannot adapt to form nonlinear features, blocking the emergence of a representation that makes the task linearly separable.

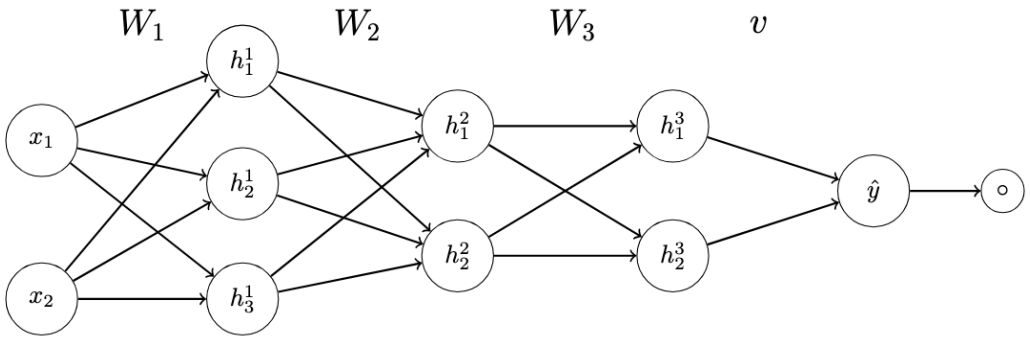
- Relate this to issues of initialization, activation choice, and depth discussed elsewhere in this session.

Initialization, activation choice, and depth jointly determine gradient stability; when gradients vanish or explode due to poor choices, early layers cannot receive meaningful learning signals, preventing the hidden representation needed to solve tasks like XOR from forming.

## Question 4

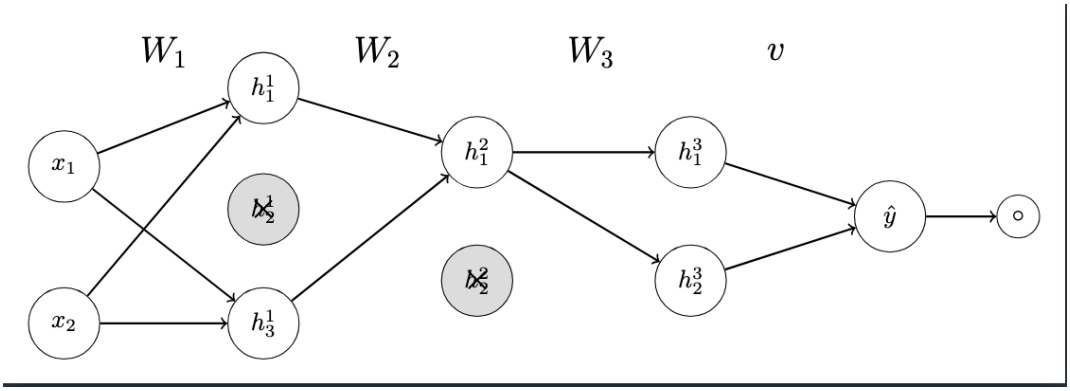
Dropout as stochastic representation perturbation

here is a basic representation of the network



a) Compute the network output for the input $x^\top = (1, 1)$ using the following dropout masks:

$$\mu^1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad \mu^2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \qquad \mu^3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$




$$W^{(1)\top} = \begin{pmatrix} 1 & 2 \\ -1 & 1 \\ 2 & -1 \end{pmatrix}, \qquad W^{(2)\top} = \begin{pmatrix} 1 & 2 & -2 \\ 2 & -1 & 1 \end{pmatrix}, \qquad W^{(3)\top} = \begin{pmatrix} 2 & 1 \\ -1 & -1 \end{pmatrix}, \qquad v^\top = \begin{pmatrix} 1 & -1 \end{pmatrix}.$$

$$b^1 = \begin{pmatrix} -1 \\ 1 \\ 2 \end{pmatrix}, \qquad b^2 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \qquad b^3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \qquad b^4 = 1.$$

### Input mask

$$\tilde{x} = \mu^1 \odot x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

### Layer 1

$$a^1 = W^{(1)\top}\tilde{x} + b^1 = \begin{pmatrix} 1 & 2 \\ -1 & 1 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}$$

$$h^1 = \mathrm{ReLU}(a^1) = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}, \qquad \tilde{h}^1 = \mu^2 \odot h^1 = \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix}.$$

### Layer 2

$$a^2 = W^{(2)\top}\tilde{h}^1 + b^2 = \begin{pmatrix} 1 & 2 & -2 \\ 2 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -4 \\ 7 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ 8 \end{pmatrix}$$

$$h^2 = \mathrm{ReLU}(a^2) = \begin{pmatrix} 0 \\ 8 \end{pmatrix}, \qquad \tilde{h}^2 = \mu^3 \odot h^2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

### Layer 3

$$a^3 = W^{(3)\top} \tilde{h}^2 + b^3 = \begin{pmatrix} 2 & 1 \\ -1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

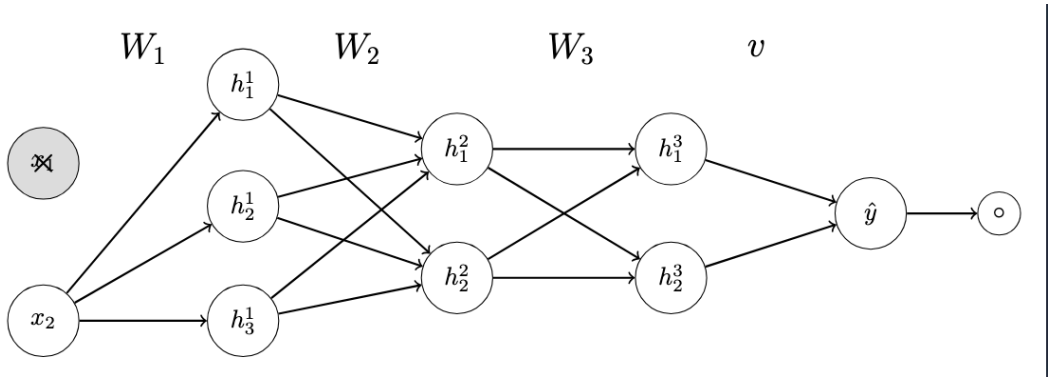
$$h^3 = \mathrm{ReLU}(a^3) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

### Output

$$a^4 = v^\top h^3 + b^4 = \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 1 = (1 - 1) + 1 = 1.$$

(b) Repeat the computation using:

$$\mu^1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad \mu^2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \qquad \mu^3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$




## Step 1. Input mask

$$\tilde{x} = \mu^1 \odot x = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

## Step 2. Layer 1

$$a^1 = W^{(1)\top} \tilde{x} + b^1 = \begin{pmatrix} 1 & 2 \\ -1 & 1 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix}.$$

ReLU:

$$h^1 = \mathrm{ReLU}(a^1) = \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix}.$$

Apply mask $\mu^2$ (all ones, nothing dropped):

$$\tilde{h}^1 = \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix}.$$

## Step 3. Layer 2

$$a^2 = W^{(2)\top}\tilde{h}^1 + b^2 = \begin{pmatrix} 1 & 2 & -2 \\ 2 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 3 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

Compute components:

$$a_1^2 = 1 + 6 - 6 + 2 = 3, \quad a_2^2 = 2 - 3 + 3 + 1 = 3.$$

$$a^2 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \qquad h^2 = \mathrm{ReLU}(a^2) = \begin{pmatrix} 3 \\ 3 \end{pmatrix}.$$

Apply mask $\mu^3$ (all ones):

$$\tilde{h}^2 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}.$$

## Step 4. Layer 3

$$a^3 = W^{(3)\top}\tilde{h}^2 + b^3 = \begin{pmatrix} 2 & 1 \\ -1 & -1 \end{pmatrix} \begin{pmatrix} 3 \\ 3 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 9 \\ -6 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 10 \\ -5 \end{pmatrix}.$$

ReLU:

$$h^3 = \begin{pmatrix} 10 \\ 0 \end{pmatrix}.$$

## Step 5. Output

$$a^4 = v^\top h^3 + b^4 = \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} 10 \\ 0 \end{pmatrix} + 1 = 10 + 1 = 11.$$

$\hat{y} = 11$

**C) Compare the two outputs and discuss:**
**• how dropout changes the effective computation graph,**
**• which features are relied upon across different masks,**

**• why dropout encourages robustness and redundancy in learned representations.**

Dropout changes the effective computation graph by randomly removing units, causing the network to compute predictions using different sub-networks for the same input. Features that consistently influence the output across different masks are therefore preferred, while reliance on any single neuron is discouraged. This forces information to be encoded redundantly across multiple units, improving robustness and reducing overfitting.

**d) Discuss how applying dropout during training affects the gradients received by individual units across iterations. In particular, comment on gradient variance and the consequences for representation learning.**

Dropout increases gradient variance by randomly zeroing units, preventing consistent co-adaptation and forcing gradients to favor features that are robust across many sub-networks, leading to more distributed and generalizable representations.

## Question 5

**a) Compare shallow and deep neural networks in terms of representational efficiency. Why can certain functions be represented more compactly using depth?**

hallow networks often require an exponential number of units to represent functions that deep networks can express with a polynomial number by composing simpler functions across layers. Depth enables hierarchical reuse of intermediate features, making representations far more compact. Certain functions are therefore efficiently representable only when structure is distributed across multiple layers.

**(b) Discuss why increasing depth generally makes optimization more difficult.**

increasing depth compounds nonlinear transformations and Jacobians, making gradients more prone to vanishing or exploding. As a result, error signals reaching early layers become unstable or uninformative, slowing or preventing learning. The loss landscape also becomes more ill-conditioned with depth.

**(c) Explain how initialization schemes, normalization methods, and optimization al- gorithms interact to make deep representation learning feasible in practice.**

Proper initialization preserves signal variance, normalization stabilizes activations and gradients during training, and modern optimizers smooth and adapt gradient updates. Together, they maintain usable gradient flow across many layers. This interaction is what allows deep networks to actually learn the representations they can theoretically express.

**(d) Summarize why depth is a conditional advantage: powerful when gradient flow is stable, and detrimental otherwise.**

Depth is beneficial only when gradients propagate stably, allowing layers to learn useful intermediate features. When gradient flow breaks down, additional layers add optimization difficulty without representational gain. Thus, depth amplifies both the power and the risks of neural networks.