Rahul Raj : NLM Notes

CSC1185 Neural Language Modelling

WEEK 1 Notes : Logistic Regression

1. IMPORTANT TOPICS LIST

Major Topics:

Logistic Regression (LR) as a probabilistic ML classifier

Generative vs. Discriminative classifiers

Components of a supervised probabilistic ML method

Binary Logistic Regression (sigmoid function)

Sentiment Classification with LR

Feature design and scaling (normalisation vs. standardisation)

Processing many examples at once (vectorised/matrix form)

Choosing a classifier (LR vs. Naive Bayes)

Multinomial Logistic Regression

Softmax function

Binary vs. Multinomial LR (architecture comparison)

Learning in Logistic Regression

Cross-Entropy Loss Function

Gradient Descent

Stochastic Gradient Descent (SGD)

Mini-batch and Batch Gradient Descent

Overfitting

Regularisation (L1 and L2)

Learning in Multinomial Logistic Regression

2. DEFINITIONS

Term	Definition
Logistic Regression (LR)	A supervised, discriminative probabilistic machine learning method for classification tasks; models the predictive power individual features have in predicting two or more possible outcomes
Binary LR	Two-class LR that predicts a single scalar, thresholded at 0.5 for class membership
Multinomial LR	Multi-class LR that predicts a vector of K scalars, one per class; the highest value indicates class membership
Generative classifier	A model that learns the distribution of each class (what examples of each class look like); can generate examples. Example: Naive Bayes
Discriminative classifier	A model that learns the boundary between classes (what distinguishes them). Example: Logistic Regression
Feature representation	A vector of real-valued features [x_1, x_2, ..., x□] computed from an input observation
Sigmoid function (σ)	A function that maps any real value z to the range (0,1); used as the activation/output function in binary LR
Softmax function	Also called softargmax or normalised exponential function; converts a vector of K real numbers into a probability distribution over K outcomes; generalisation of sigmoid to multiple dimensions

Cross-entropy loss (LCE)	The negative log probability of the correct label; the loss function used in logistic regression and neural networks
Gradient descent	An optimisation algorithm that minimises a loss function by iteratively moving in the direction opposite to the gradient (steepest slope)
Stochastic Gradient Descent (SGD)	Gradient descent where weight updates are computed for each individual training instance
Mini-batch gradient descent	Gradient descent where weight updates are computed over batches of m training instances
Batch gradient descent	Gradient descent where m equals the entire training set size
Overfitting	When a model learns the training data too well (including noise), performing excellently on training but poorly on unseen data
Regularisation	Adding a penalty term $R(\theta)$ to the cost function to constrain model complexity and encourage generalisation
L1 regularisation (Lasso)	Adds a penalty based on the absolute values of weights (Manhattan distance from origin); can shrink some weights to exactly zero — performs feature selection
L2 regularisation (Ridge)	Adds a penalty based on the square of weights (Euclidean distance from origin); reduces all weights but keeps them non-zero
Gold label (y)	The correct/true output label for a training example
$\hat{y}$ (y-hat)	The classifier's estimated/predicted output

Bias term (b)	An offset added to the weighted sum; can be implemented as a weight w_0 with input value 1
Learning rate (η)	A hyperparameter controlling the step size during gradient descent
One-hot vector	A vector where one element is 1 (the correct class) and all others are 0; used in multinomial LR gold labels
Feature normalisation	Scaling features to range [0,1] using min and max values
Feature standardisation	Scaling features to have mean 0 and standard deviation 1
Conditional maximum likelihood estimation	Choosing parameters w, b that maximise the log probability of the true y labels given observations x
Loss function L(ŷ, y)	A function expressing how much the classifier's output ŷ differs from the correct output y
Model parameters θ	The set of weights w and bias b: θ = {w, b}
Period disambiguation	The NLP task of deciding whether a period in text indicates end-of-sentence or not
Feature templates	Automatic methods to create large numbers of features, e.g., 2-gram templates

---

3. KEY CONCEPTS EXPLAINED

3.1 LR as a Probabilistic ML Classifier

A supervised probabilistic ML method requires four components:

Feature representation — each input $x^{(i)}$ is represented as a vector $[x_1, x_2, ..., x_n]$

Classification function — computes $\hat{y}$ via $p(y|x)$; uses sigmoid or softmax

Objective function — cross-entropy loss to measure error

Optimisation algorithm — stochastic gradient descent to minimise loss

Training phase: Optimise weights **w** and bias *b* using SGD and cross-entropy loss.
Application: Given input x, compute $p(y|x)$ and return the higher-probability class ($y=1$ or $y=0$).

3.2 Generative vs. Discriminative Classifiers

Property	**Generative (Naive Bayes)**	**Discriminative (Logistic Regression)**
What it models	Distribution of each class	Decision boundary between classes
Can generate examples?	Yes	No
Dog/cat analogy	"What do dogs look like?"	"What separates dogs from cats?"
Handles feature correlation	Poorly (treats as independent)	Better (distributes weight)
Performance on small datasets	Sometimes better	Generally worse
Performance on large datasets	Generally worse	Generally better

---

3.3 The Sigmoid Function

The sigmoid squashes the weighted sum z (which ranges $-\infty$ to $+\infty$) into the range (0,1), making it interpretable as a probability.

Decision threshold: $\sigma(z) = 0.5 \rightarrow$ classify as y=1 if output > 0.5, else y=0

$P(y=0|x) = 1 - P(y=1|x)$ — the two probabilities are complementary

Also called the **logistic function** — gives logistic regression its name

The slope is steepest at the decision boundary (z=0), which has mathematical advantages for learning

---

3.4 Sentiment Classification Example

Given movie review text with features:

x_1 = count of positive lexicon words

x_2 = count of negative lexicon words

x_3 = 1 if "no" $\in$ doc, else 0

x_4 = count of 1st and 2nd person pronouns

x_5 = 1 if "!" $\in$ doc, else 0

x_6 = ln(word count of doc)

With weights w = [2.5, −5.0, −1.2, 0.5, 2.0, 0.7] and bias b = 0.1:

w_1 = 2.5 (positive) → positive words push prediction toward positive class

w_2 = −5.0 (negative, larger magnitude) → negative words are about twice as important as positive words in influencing the decision

---

3.5 Feature Design and Scaling

Both Naive Bayes and LR require **hand-designed features**

Neither can model **feature interactions** directly — interactions must be encoded as dedicated features

NLP is moving toward **automatic feature learning**

Large feature sets can be created via **feature templates** (e.g., 2-gram templates)

Scaling is important for LR because features with different ranges can cause unequal gradient updates

---

3.6 Vectorised Processing (Many Examples at Once)

Rather than looping over examples, matrix operations allow computing outputs for all m examples simultaneously:

Matrix X: size [m × f] — m examples, each with f features **Weight matrix w:** size [f × 1] **Bias b:** size [m × 1] — same bias value repeated m times **Output y:** size [m × 1]

This exploits modern hardware's ability to perform matrix operations very efficiently (GPUs etc.), which is critical for large datasets.

3.7 Choosing Between LR and Naive Bayes

LR generally works better on larger documents or datasets

LR handles feature correlations better: assigns partial weight to correlated features, while Naive Bayes treats them as independent and doubles-counts evidence

Naive Bayes works better on very small datasets or short documents

Naive Bayes is faster to train (no optimisation step) — still reasonable in some situations

---

3.8 Multinomial LR and Softmax

Extends binary LR to K classes (hard classification — only one class can be correct)

Output is a vector $\hat{\mathbf{y}}$ of length K where $\hat{y}_k = p(y_k=1|x)$

The class with highest $\hat{y}_k$ is the predicted class

Requires separate weight vector $\mathbf{w}_k$ and bias b_k for each of the K classes

Weight vectors are packed into weight matrix **W** of size [K × f]

3.9 Cross-Entropy Loss — Derivation Intuition

Goal: maximise the probability of the correct label → **conditional maximum likelihood estimation**

Since maximising p is equivalent to maximising log p, use log probabilities (mathematically convenient — products become sums)

Flip the sign to convert maximisation into minimisation → **negative log likelihood = cross-entropy loss**

The loss surface is **convex** → gradient descent is guaranteed to find the global optimum (provided learning rate is sufficiently small)

Why CE loss works:

When $\hat{y} = 1$ and $y = 1$: $-\log(1) = 0$ → zero loss ✓

When $\hat{y} = 0$ and $y = 1$: $-\log(0) = \infty$ → infinite loss ✓

Maximising probability of correct class simultaneously minimises probability of incorrect class (they sum to 1)

3.10 Gradient Descent — Conceptual Understanding

Analogy: Finding the bottom of a valley in dense fog without a map — you feel for the slope in every direction and take a step downhill.

The **gradient vector g** has one element per parameter (each weight and bias)

Each element is the **partial derivative** of the loss with respect to that parameter

Tells us whether to increase or decrease each weight to reduce loss

Stochastic GD (m=1): erratic movements through parameter space

Mini-batch GD: smoother gradient estimates

Batch GD (m=all data): stable but slow per update

3.11 Overfitting

Two key causes:

Model complexity — too many parameters relative to number of observations

Insufficient/non-diverse training data — model memorises quirks rather than patterns

Example: A 4-gram model on tiny data → 100% training accuracy but fails on unseen 4-grams.

Techniques to avoid overfitting:

Regularisation (L1 or L2)

Early stopping — stop training when validation performance degrades

Cross-validation — k-fold assessment on different data subsets

---

3.12 L1 vs. L2 Regularisation

Property	**L1 (Lasso)**	**L2 (Ridge)**
Penalty	Sum of absolute values	Sum of squares
Norm used	Manhattan (L1)	Euclidean (L2)
Effect on weights	Can shrink to exactly 0	Shrinks all but rarely to 0
Feature selection	Yes (sparse solution)	No
Derivative	Non-continuous at 0	Simple: 2θ
Easier to optimise	No	Yes
Preference	Few large weights, many zeros	Many small weights
Bias term	Generally ignored	Generally ignored

---

4. IMPORTANT FORMULAS AND EQUATIONS

Weighted sum (z):

Iterative: $z = (\Sigma_{i=1}^{n} w_i x_i) + b$

Vectorised: $z = \mathbf{w} \cdot \mathbf{x} + b$

Sigmoid function:

$\sigma(z) = 1 / (1 + e^{-z}) = 1 / (1 + \exp(-z))$

Input: $z \in (-\infty, +\infty)$

Output: $\sigma(z) \in (0, 1)$

Decision threshold: $\sigma(z) = 0.5$ (i.e., $z = 0$)

Binary LR output:

$P(y=1|x) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$

$P(y=0|x) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b)$

Matrix multiplication (single example):

$[w_0\ w_1\ ...\ w\square] \times [1, x_1, x_2, ..., x\square]^T = [z]$

Note: Shape $[1\times(n+1)] \times [(n+1)\times1] = [1\times1]$

Matrix form (many examples):

$\mathbf{y} = \mathbf{X}^{****}\mathbf{w} + \mathbf{b}$

Shapes: $(m\times1) = (m\times f)(f\times1) + (m\times1)$

where m = number of examples, f = number of features

Softmax function:

$\text{softmax}(\mathbf{z}_i) = \exp(\mathbf{z}_i) / \Sigma\square_{=1}^{K} \exp(\mathbf{z}\square)$ for $1 \le i \le K$

Full output vector: $\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$

where $\mathbf{W}$ is $[K \times f]$

Softmax per-class probability:

$p(\hat{y}_k=1|\mathbf{x}) = \exp(\mathbf{w}_k \cdot \mathbf{x} + b_k) / \sum_{k=1}^{K} \exp(\mathbf{w}_k \cdot \mathbf{x} + b_k)$

Bernoulli probability mass function:

$p(y|x) = \hat{y}^y(1-\hat{y})^{1-y}$ for $y \in \{0,1\}$

Log probability:

$\log p(y|x) = y \log \hat{y} + (1-y) \log(1-\hat{y})$

Cross-entropy loss (binary):

$L\_CE(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$

Special cases:

$y = 1$: $L\_CE = -\log(\hat{y})$

$y = 0$: $L\_CE = -\log(1-\hat{y})$

Optimal parameters:

$\theta' = \text{argmin}_\theta \frac{1}{m} \sum_{i=1}^{m} L\_CE(f(x^{(i)}; \theta), y^{(i)})$

where $f(x^{(i)}, \theta) = \sigma(\mathbf{w} \cdot x^{(i)} + b)$

Partial derivative of cross-entropy loss w.r.t. weight w_k:

$\partial L\_CE(\hat{y},y) / \partial w_k = [\sigma(\mathbf{w} \cdot \mathbf{x}+b) - y]x_k = (\hat{y} - y)x_k$

Parameter update rule (gradient descent):

$\theta^{t+1} = \theta^t - \eta \nabla L(f(x; \theta), y)$

where η = learning rate, ∇L = gradient vector

Mini-batch gradient (iterative):

$\partial L\_CE(\hat{y},y) / \partial w_j = (1/m) \Sigma_{i=1}^{m} [\sigma(\mathbf{w} \cdot x^{(i)} + b) - y^{(i)}] x_j^{(i)}$

Mini-batch gradient (matrix form):

$\partial L\_CE(\hat{y},y) / \partial \mathbf{w} = (1/m)(\hat{y} - \mathbf{y})^T \mathbf{X} = (1/m)(\sigma(\mathbf{X}\mathbf{w} + \mathbf{b}) - \mathbf{y})^T \mathbf{X}$

Feature normalisation:

$\mathbf{x'}_i = (\mathbf{x}_i - \min(\mathbf{x}_i)) / (\max(\mathbf{x}_i) - \min(\mathbf{x}_i))$

Feature standardisation:

$\mu_i = (1/m) \Sigma_{j=1}^{m} x_i^{(j)}$

$\sigma_i = \sqrt{[(1/m) \Sigma_{j=1}^{m} (x_i^{(j)} - \mu_i)^2]}$

$\mathbf{x'}_i = (\mathbf{x}_i - \mu_i) / \sigma_i$

L2 regularisation objective:

$\theta^* = \text{argmax}_\theta [\Sigma_{i=1}^{m} \log P(y^{(i)}|x^{(i)})] - \alpha \Sigma_{j=1}^{n} \theta_j^2$

where $R(\theta) = \|\theta\|_2^2 = \Sigma_j \theta_j^2$

L1 regularisation objective:

$\theta^* = \text{argmax}_\theta [\Sigma_{i=1}^{m} \log P(y^{(i)}|x^{(i)})] - \alpha \Sigma_{j=1}^{n} |\theta_j|$

where $R(\theta) = \|\theta\|_1 = \Sigma_i |\theta_i|$

Cross-entropy loss (multinomial):

$L\_CE(\hat{\mathbf{y}}, \mathbf{y}) = -\Sigma_{k=1}^{K} y_k \log \hat{y}_k = -\log \hat{y}\_c = -\log p(y\_c=1|\mathbf{x})$

$= -\log [\exp(\mathbf{w\_c}\cdot\mathbf{x}+b\_c) / \Sigma_{k=1}^{K} \exp(\mathbf{w\_j}\cdot\mathbf{x}+b_k)]$

where c is the correct class

Gradient in multinomial LR:

$\partial L\_CE / \partial w\_\{k,i\} = -(y_k - \hat{y}_k)x_i = -(y_k - p(y_k=1|\mathbf{x}))x_i$

---

5. THINGS TO REMEMBER

Important Constraints & Facts

Sigmoid output is **strictly between (0,1)**, never 0 or 1 exactly

Classification threshold is **0.5** (not adjustable in basic LR)

The bias term b can be **absorbed into the weight vector** as w_0 with input value 1

In CS, **dot product notation (·) is used for matrix multiplication**

The cross-entropy loss surface for LR is **convex** — gradient descent is guaranteed to find the global optimum

Normally weight vector **w** is assumed to be n×1 shape; transpose $\mathbf{w}^T$ indicates 1×n shape

Common Misconceptions

❌ "Logistic regression is a generative model" → **NO**, it is discriminative

❌ "Sigmoid and softmax are unrelated" → Sigmoid is a **special case of softmax** with K=2

❌ "Larger weights always mean better models" → **NO**, large weights can cause overfitting; regularisation penalises them

❌ "L1 and L2 both shrink weights toward zero" → **L2 rarely reaches exactly zero; L1 can produce exactly zero weights**

❌ "SGD with m=full dataset is SGD" → **NO**, that is batch gradient descent; SGD uses m=1

❌ "Naive Bayes always performs worse than LR" → **NO**, NB can outperform LR on very small datasets or short documents

❌ "Regularisation includes the bias term" → For both L1 and L2, **the bias is generally ignored**

Tricky Points Professors Test

The weighted sum z can range $-\infty$ **to** $+\infty$ — this is WHY sigmoid is needed

P(y=0|x) = 1 − P(y=1|x) — they are **not independently computed**

The partial derivative $\partial L\_CE/\partial w_{\square}$ = **(ŷ − y)x**$_{\square}$ — this is elegantly simple and important to know

Negative words (w_2=−5) have **twice the magnitude** of positive words (w_1=2.5) in the example — sign matters for direction, magnitude matters for importance

Cross-entropy loss example: y=1, ŷ=0.7 → L_CE = **0.36**; y=0, ŷ=0.7 → L_CE = **1.2** (much higher loss for wrong label)

Mini-batch with m=1 IS stochastic GD; mini-batch with m=all data IS batch GD

Feature interactions **cannot** be modelled by LR or NB directly — must be encoded as new features

In the matrix equation **y = Xw + b**, the shapes must be: X is [m×f], w is [f×1], b is [m×1], y is [m×1]

Softmax is also called **softargmax** or **normalised exponential function** — know all three names

LR is the **foundation of neural networks** — a neural network can be viewed as a **stack of logistic regression classifiers**

α in the regularisation objective is the **regularisation strength hyperparameter**

Model Assumptions

Binary LR assumes only two possible output classes

Multinomial LR assumes **hard classification** — one item cannot belong to multiple classes

Gold labels in multinomial LR are **one-hot vectors**

The loss surface being convex depends on the cross-entropy loss + sigmoid combination

Parameter Ranges

σ(z) ∈ **(0, 1)** — open interval, never exactly 0 or 1

Softmax outputs ∈ **[0,1]** and sum to exactly **1**

Learning rate η: small enough to ensure convergence; if too large, may not converge

Regularisation strength α: higher = more regularisation = simpler model

6. CODE-RELATED CONCEPTS

Note: Week 1 PDF does not contain explicit PyTorch code snippets. However, the following implementation concepts are discussed and are testable:

Matrix Multiplication Shape Rules

```
None


# Binary LR — single example

# w shape: [1 × (n+1)], x shape: [(n+1) × 1] → z shape: [1 × 1]

z = w @ x  # dot product




# Many examples

# X shape: [m × f], w shape: [f × 1], b shape: [m × 1]

# y shape: [m × 1]

y = X @ w + b

sigmoid_y = 1 / (1 + exp(-y))
```

Conceptual Implementation Notes

Bias implementation: Can be treated as weight w_0 with a constant input of 1 appended to x

Softmax implementation: Uses exp() of each element divided by sum of all exp() values — numerically stable implementations use log-softmax in practice

Why matrix operations? Exploits GPU-level parallelism; much faster than for-loops over individual examples

SGD update: $\theta = \theta - \eta \times$ gradient (subtract because minimising)

Gradient of CE loss: Elegantly simplifies to $(\hat{y} - y) \times x_{\square}$ for each weight $w_{\square}$

Vectorised vs. Iterative

```
None

# ITERATIVE (slow):

for x_i in training_set:

    y_pred = sigmoid(w @ x_i + b)



# VECTORISED (fast, preferred):

Y_pred = sigmoid(X @ w + b)  # computes all at once
```

Feature Normalisation vs Standardisation (Implementation)

```
None

# Normalisation (min-max scaling):

x_norm = (x - x.min()) / (x.max() - x.min())

# Output range: [0, 1]



# Standardisation (z-score):

mu = x.mean()

sigma = x.std()

x_std = (x - mu) / sigma

# Output: mean=0, std=1
```

End of Week 1 Notes. Ready for Part 2 — MCQs when you ask.

WEEK 1 — MCQs

Section A — Questions

Q1. Logistic Regression is best described as:

A) A generative probabilistic classifier B) A discriminative probabilistic classifier C) An unsupervised learning method D) A rule-based classification system

---

Q2. Which of the following is NOT one of the four components of a supervised probabilistic ML method?

A) A feature representation B) A classification function C) A data augmentation strategy D) An objective function for learning

---

Q3. The sigmoid function $\sigma(z)$ maps its input to which range?

A) $[-1, 1]$ B) $[0, 1]$ C) $(0, 1)$ D) $(-\infty, +\infty)$

Q4. What is the decision threshold used in binary logistic regression?

A) 0.0 B) 1.0 C) 0.5 D) 0.25

Q5. Which of the following correctly defines $P(y=0|x)$ in binary LR?

A) $\sigma(w \cdot x + b)$ B) $1 + \sigma(w \cdot x + b)$ C) $1 - \sigma(w \cdot x + b)$ D) $\sigma(-w \cdot x + b)$ only

---

Q6. In the sentiment classification example, weights w = [2.5, −5.0, −1.2, 0.5, 2.0, 0.7]. What does the magnitude of $w_2 = -5.0$ tell us compared to $w_1 = 2.5$?

A) Positive words are twice as important as negative words B) Negative words are about twice as important as positive words C) Both features are equally important D) Negative words have no impact on classification

---

Q7. The sigmoid function is also known as:

A) The softmax function B) The logistic function C) The ReLU function D) The normalised exponential function

---

Q8. Which formula correctly represents the sigmoid function?

A) $\sigma(z) = z / (1 + z)$ B) $\sigma(z) = 1 / (1 - e^{-z})$ C) $\sigma(z) = 1 / (1 + e^{-z})$ D) $\sigma(z) = e\text{^}z / (1 + e\text{^}z^2)$

Q9. The vectorised definition of the weighted sum z is:

A) $z = w + x + b$ B) $z = w \cdot x + b$ C) $z = w \times x^2 + b$ D) $z = \sigma(w \cdot x)$

Q10. Why is the sigmoid function applied to $z = w \cdot x + b$?

A) Because z already outputs probabilities B) Because z ranges from $-\infty$ to $+\infty$ and needs to be mapped to (0,1) C) Because z is always negative D) Because sigmoid increases training speed

---

Q11. In the sentiment example with x = [3,2,1,3,0,4.19], w = [2.5,−5.0,−1.2,0.5,2.0,0.7], b=0.1, what is P(y=1|x)?

A) 0.3 B) 0.5 C) 0.833 D) 0.7

Q12. In the same example above, what is P(y=0|x)?

A) 0.7 B) 0.3 C) 0.5 D) 0.833

Q13. Naive Bayes is classified as which type of classifier?

A) Discriminative B) Generative C) Probabilistic-discriminative D) Unsupervised

---

Q14. How does Logistic Regression handle correlated features compared to Naive Bayes?

A) LR also treats them as independent B) LR ignores one of the correlated features entirely C) LR distributes weight between correlated features D) LR multiplies both correlated features together

---

Q15. Naive Bayes tends to perform better than LR when:

A) The dataset is very large B) The documents are very long C) The dataset is very small or documents are short D) Features are highly correlated

---

Q16. Feature normalisation transforms features to which range?

A) $(-1, 1)$ B) $[0, 1]$ C) Mean=0, Std=1 D) $(0, \infty)$

Q17. Feature standardisation produces output with:

A) Values between 0 and 1 B) Mean = 0 and standard deviation = 1 C) Values between −1 and 1 D) All positive values

---

Q18. The formula for feature normalisation is:

A) $x'_i = (x_i - \mu_i) / \sigma_i$ B) $x'_i = x_i / \max(x_i)$ C) $x'_i = (x_i - \min(x_i)) / (\max(x_i) - \min(x_i))$ D) $x'_i = x_i - \text{mean}(x_i)$

Q19. When processing m examples at once using matrix operations, if X is [m×f] and w is [f×1], the output y has shape:

A) [f×1] B) [m×m] C) [m×1] D) [1×f]

Q20. Why is vectorised processing (matrix operations) preferred over for-loops?

A) It requires less memory B) It exploits modern computing hardware's ability to perform matrix operations efficiently C) It gives more accurate results mathematically D) It avoids the need for a loss function

---

Q21. In multinomial logistic regression, what does the output vector $\hat{y}$ represent?

A) A single binary probability B) A vector of K probabilities, one per class C) The raw weighted sums before activation D) The gradient vector

---

Q22. What function is used as the output activation in multinomial logistic regression?

A) Sigmoid B) ReLU C) Softmax D) Tanh

Q23. Softmax is also known as: (select the most complete answer)

A) Normalised sigmoid B) Softargmax or normalised exponential function C) Hard-max function D) Logistic exponential

---

Q24. The softmax formula for element i of vector z is:

A) $\exp(z_i) / K$ B) $z_i / \Sigma\square\, z\square$ C) $\exp(z_i) / \Sigma\square_{=1}^{K} \exp(z\square)$ D) $1 / (1 + \exp(-z_i))$

Q25. In multinomial LR, how many weight vectors are needed for K classes?

A) 1 weight vector shared across all classes B) K separate weight vectors (one per class) C) K−1 weight vectors D) 2 weight vectors regardless of K

Q26. Gold labels in multinomial LR are represented as:

A) A scalar value between 0 and K B) A probability distribution C) A one-hot vector D) A softmax output

---

Q27. A neural network can be viewed as:

A) A single logistic regression classifier B) A stack of logistic regression classifiers C) A stack of Naive Bayes classifiers D) A single softmax layer

---

Q28. The cross-entropy loss function is derived from:

A) Mean squared error B) Maximising the log probability of correct labels (conditional maximum likelihood estimation) C) Minimising the absolute difference between $\hat{y}$ and y D) Maximising the distance between class probabilities

---

Q29. The cross-entropy loss for a single example with y=1 simplifies to:

A) $-\log(1 - \hat{y})$ B) $-\log(\hat{y})$ C) $\log(\hat{y})$ D) $\hat{y} - y$

Q30. The cross-entropy loss for a single example with y=0 simplifies to:

A) $-\log(\hat{y})$ B) $\log(1 - \hat{y})$ C) $-\log(1 - \hat{y})$ D) $\hat{y}$

Q31. Using the example from the slides: $\hat{y} = 0.7$ and $y = 1$. What is L_CE?

A) 0.7 B) 1.2 C) 0.36 D) 0.3

Q32. Using the same example: $\hat{y} = 0.7$ and $y = 0$. What is L_CE?

A) 0.36 B) 0.7 C) 1.2 D) 0.3

Q33. Why are log probabilities used in deriving the cross-entropy loss?

A) Logs make the computation faster B) Whatever values maximise a probability also maximise its log — mathematically convenient C) Log probabilities are always positive D) To convert multiplication into exponentiation

---

Q34. The loss surface of logistic regression with cross-entropy loss is:

A) Non-convex with many local minima B) Convex — gradient descent finds the global optimum C) Concave — gradient ascent is used D) Flat — no gradient information available

---

Q35. The partial derivative of L_CE with respect to weight w_j in logistic regression is:

A) $(y - \hat{y})x_j$ B) $(\hat{y} - y)$ C) $(\hat{y} - y)x_j$ D) $\sigma(w_j)x_j$

Q36. The parameter update rule in gradient descent is:

A) $\theta^{t+1} = \theta^t + \eta \nabla L$ B) $\theta^{t+1} = \theta^t - \eta \nabla L$ C) $\theta^{t+1} = \theta^t \times \eta \nabla L$ D) $\theta^{t+1} = \eta - \theta^t \nabla L$

Q37. In the gradient descent walk-through example with initial $\theta^0 = [0,0,0]$ and $\eta=0.1$, the gradient was $[-1.5, -1.0, -0.5]$. What are the updated weights θ^1?

A) $[-0.15, -0.10, -0.05]$ B) $[1.5, 1.0, 0.5]$ C) $[0.15, 0.10, 0.05]$ D) $[0.0, 0.0, 0.0]$

Q38. Stochastic Gradient Descent (SGD) refers to:

A) Computing gradients over the full training set B) Computing gradients over random mini-batches C) Computing and updating weights for each individual training instance D) Computing gradients using second-order derivatives

---

Q39. If m = size of the entire training set in mini-batch gradient descent, this becomes:

A) Stochastic gradient descent B) Batch gradient descent C) Mini-batch gradient descent D) Online gradient descent

---

Q40. Which of the following best describes overfitting?

A) The model performs well on both training and test data B) The model performs poorly on training data but well on test data C) The model learns training data too well including noise, performing poorly on unseen data D) The model has too few parameters to learn patterns

---

Q41. Which technique can perform automatic feature selection during regularisation?

A) L2 regularisation (Ridge) B) Early stopping C) L1 regularisation (Lasso) D) Cross-validation

---

Q42. L2 regularisation penalises weights based on:

A) The absolute values of the weights B) The square of the weights C) The log of the weights D) The number of non-zero weights

---

Q43. Which regularisation method is easier to optimise and why?

A) L1, because its derivative is non-continuous B) L2, because its derivative is simply 2θ C) L1, because it produces sparse solutions D) L2, because it sets weights to exactly zero

---

Q44. The regularisation term R(θ) is applied to:

A) Only the bias term B) Both weights and bias equally C) Only the weights (bias is generally ignored) D) Only the largest weight

---

Q45. The mini-batch gradient in matrix form is:

A) $(1/m)(\hat{y}-y)X$ B) $(1/m)(\hat{y}-y)^TX$ C) $(\hat{y}-y)^TX$ D) $(1/m)X^T(\hat{y}-y)$

Q46. In multinomial LR, the cross-entropy loss simplifies to:

A) $-\Sigma\square \log \hat{y}\square$ B) $-\log \hat{y}\_c$ where c is the correct class C) $\Sigma\square\ y\square \log \hat{y}\square$ D) $-\log(1-\hat{y}\_c)$

Q47. Which of the following is TRUE about the softmax function outputs?

A) They can be negative B) They sum to more than 1 C) Each value is in [0,1] and all values sum to 1 D) They are identical to sigmoid outputs

---

Q48. The bias term b in logistic regression can be implemented as:

A) A separate learning rate B) A weight w_0 with a constant input value of 1 C) The average of all weights D) An additional feature computed from x

---

Q49. Neither Naive Bayes nor Logistic Regression can model:

A) Binary classification B) Feature interactions directly — these must be encoded as dedicated features C) Text classification D) Probability outputs

---

Q50. Which statement about the gradient vector is correct?

A) It has one element per training example B) It has one element per layer in the network C) Each element is the partial derivative of the loss w.r.t. one parameter D) It is always a scalar value

---

Q51. In the period disambiguation task, which feature would suggest the period is NOT end-of-sentence?

A) $x_1 = 1$ if $\text{Case}(w_i) = \text{Lower}$ B) $x_2 = 1$ if $w_i \in \text{AcronymDict}$ C) A space following the period D) The word before the period is in lowercase

---

Q52. The model parameters θ in logistic regression refer to:

A) Only the weights w B) Only the bias b C) Both weights w and bias b D) The learning rate and batch size

---

Q53. TRUE or FALSE: Increasing regularisation strength α always improves test performance.

A) True — more regularisation always helps B) False — too much regularisation underfits the data C) True — regularisation eliminates all overfitting D) False — regularisation has no effect on test performance

---

Q54. The gradient of the loss is described as pointing in the direction of:

A) The minimum of the loss function B) The steepest ascent of the loss function C) The steepest descent of the loss function D) The average loss across all examples

---

Q55. In the weight matrix **W** for multinomial LR with K classes and f features, what is the shape of W?

A) [f × K] B) [K × f] C) [f × f] D) [K × K]

Q56. Which of the following correctly orders the steps in training logistic regression?

A) Compute loss → Update weights → Compute output → Compute gradient B) Compute output → Compute loss → Compute gradient → Update weights C) Update weights → Compute gradient → Compute loss → Compute output D) Compute gradient → Compute output → Update weights → Compute loss

---

Q57. What does the notation $f(x^{(i)}; \theta)$ represent in the gradient descent formulation?

A) The loss function value B) The model output $\hat{y}$ computed from input $x^{(i)}$ and parameters θ C) The gradient vector D) The regularisation penalty

---

Q58. Which property of the sigmoid function is described as "desirable" in the slides?

A) Its output is always greater than 0.5 B) Its slope is steepest around the decision boundary C) It produces exactly 0 or 1 at extremes D) It is computationally cheaper than softmax

---

Q59. L1 regularisation is said to prefer:

A) Many small but non-zero weights B) All weights being equal C) Sparse solutions — few large weights, many zero weights D) Weights that are normally distributed

---

Q60. A 4-gram model trained on tiny data achieves 100% accuracy on training data but fails on test data. This is an example of:

A) Underfitting B) Overfitting C) Good generalisation D) Regularisation working correctly

---

Section B — Answers & Explanations

Q1. Answer: B LR is a discriminative classifier — it models the decision boundary between classes, not the distribution of each class.

Q2. Answer: C The four components are: feature representation, classification function, objective function, and optimisation algorithm. Data augmentation is not one of them.

Q3. Answer: C $\sigma(z) \in (0,1)$ — open interval. The sigmoid never actually reaches exactly 0 or 1.

Q4. Answer: C The decision threshold is 0.5. If $\sigma(z) > 0.5$, classify as y=1; otherwise y=0.

Q5. Answer: C $P(y=0|x) = 1 - P(y=1|x) = 1 - \sigma(w \cdot x+b)$. The two probabilities are complementary.

Q6. Answer: B $|w_2| = 5.0$ vs $|w_1| = 2.5$ — negative words have roughly twice the magnitude, meaning they have twice the influence on the decision.

Q7. Answer: B The sigmoid function is also called the logistic function — which is why the method is called logistic regression.

Q8. Answer: C $\sigma(z) = 1/(1+e^{-z})$. Option D is wrong due to z^2 in the exponent.

Q9. Answer: B $z = \mathbf{w} \cdot \mathbf{x} + b$ is the vectorised weighted sum. The dot product combines all weighted features plus the bias.

Q10. Answer: B z ranges from $-\infty$ to $+\infty$ and cannot directly represent probabilities. Sigmoid maps it to (0,1).

Q11. Answer: D From the slides: $\sigma(0.833) = 0.7$. The dot product of w and x plus b = 0.833.

Q12. Answer: B $P(y=0|x) = 1 - P(y=1|x) = 1 - 0.7 = 0.3$.

Q13. Answer: B Naive Bayes is a generative classifier — it models what each class looks like and could theoretically generate examples.

Q14. Answer: C LR distributes weight between correlated features. Naive Bayes treats them as independent and multiplies both in, overestimating the evidence.

Q15. Answer: C Per the slides (citing Ng & Jordan 2002; Wang & Manning 2012): NB works better on very small datasets or short documents.

Q16. Answer: B Feature normalisation maps to [0,1] using: $x'=(x-\min)/(\max-\min)$.

Q17. Answer: B Feature standardisation (z-score) produces mean=0, standard deviation=1.

Q18. Answer: C $x'_i = (x_i - \min(x_i)) / (\max(x_i) - \min(x_i))$. Option A is standardisation, not normalisation.

Q19. Answer: C $(m \times f) \times (f \times 1) = (m \times 1)$. The output y has one value per example.

Q20. Answer: B The slides explicitly state: matrix operations exploit modern computing resources' ability to perform them efficiently, critical for large datasets.

Q21. Answer: B $\hat{y}$ is a vector of K probabilities, where $\hat{y}_k = p(y_k=1|x)$ for each class k.

Q22. Answer: C Multinomial LR uses softmax — sigmoid only works for binary classification.

Q23. Answer: B The slides explicitly name softmax as "softargmax or normalised exponential function."

Q24. Answer: C $\text{softmax}(z_i) = \exp(z_i) / \sum_{j=1}^{K} \exp(z_j)$. Option D is the sigmoid formula.

Q25. Answer: B One weight vector $\mathbf{w}_k$ and one bias b_k are needed per class, giving K weight vectors total.

Q26. Answer: C Gold labels in multinomial LR are one-hot vectors: 1 for the correct class, 0 for all others.

Q27. Answer: B Directly from the slides: "a neural network can be viewed as a stack of logistic regression classifiers."

Q28. Answer: B Cross-entropy loss comes from conditional maximum likelihood estimation — choosing parameters that maximise log probability of true labels, then flipping the sign.

Q29. Answer: B When y=1: $L_{CE} = -[1 \cdot \log(\hat{y}) + 0 \cdot \log(1-\hat{y})] = -\log(\hat{y})$.

Q30. Answer: C When y=0: $L_{CE} = -[0 \cdot \log(\hat{y}) + 1 \cdot \log(1-\hat{y})] = -\log(1-\hat{y})$.

Q31. Answer: C $L_{CE} = -\log(0.7) \approx 0.36$. This is the loss when the model correctly assigns high probability to the positive class.

Q32. Answer: C $L_{CE} = -\log(1-0.7) = -\log(0.3) \approx 1.2$. Higher loss because the model was wrong — it predicted 0.7 for the positive class but gold label was 0 (negative).

Q33. Answer: B Log is a monotonic function — maximising log p is equivalent to maximising p. It also converts products into sums, which is more computationally convenient.

Q34. Answer: B The CE loss surface is convex, so gradient descent is guaranteed to find the global optimum (given a sufficiently small learning rate).

Q35. Answer: C $\partial L_{CE}/\partial w_{\square} = (\hat{y} - y)x_{\square}$. Note the order: ŷ minus y, not y minus ŷ. This is a commonly tested detail.

Q36. Answer: B $\theta^{t+1} = \theta^t - \eta \nabla L$. We subtract because we are minimising (moving opposite to the gradient direction).

Q37. Answer: C $\theta^1 = [0,0,0] - 0.1 \times [-1.5,-1.0,-0.5] = [0,0,0] + [0.15,0.10,0.05] = [0.15, 0.10, 0.05]$.

Q38. Answer: C SGD updates weights after each individual training example (m=1).

Q39. Answer: B When m = full training set size, mini-batch gradient descent becomes batch gradient descent.

Q40. Answer: C Overfitting = excellent training performance, poor generalisation to unseen data, because the model memorised noise along with patterns.

Q41. Answer: C L1 (Lasso) can shrink coefficients to exactly zero — effectively removing features. L2 reduces weights but keeps them non-zero.

Q42. Answer: B L2 penalises the square of weights: $R(\theta) = ||\theta||_2^2 = \Sigma\theta\square^2$.

Q43. Answer: B L2 is easier to optimise because the derivative of θ^2 is simply 2θ. L1's derivative is non-continuous at zero, making it harder.

Q44. Answer: C For both L1 and L2, the bias term is generally ignored — only the weights are regularised.

Q45. Answer: B $\partial L\_CE/\partial \mathbf{w} = (1/m)(\hat{y}-\mathbf{y})^T\mathbf{X}$. The transpose is needed for the matrix dimensions to be compatible.

Q46. Answer: B Since gold labels are one-hot (all zeros except the correct class c), the sum collapses to $-\log \hat{y}_c$.

Q47. Answer: C Softmax outputs are in [0,1] and always sum to exactly 1, forming a valid probability distribution.

Q48. Answer: B The bias b can be absorbed as weight w_0 with a constant input of 1, as shown in the matrix notation in the slides.

Q49. Answer: B Both methods cannot model feature interactions directly. If an interaction is suspected, it must be manually encoded as a new dedicated feature.

Q50. Answer: C The gradient vector has one element per parameter (each weight + bias), and each element is the partial derivative of the loss w.r.t. that parameter.

Q51. Answer: B $x_2 = 1$ if $w_i \in$ AcronymDict — being in an acronym dictionary suggests the period follows an abbreviation, not an end-of-sentence.

Q52. Answer: C $\theta = \{w, b\}$ — both the weight vector and the bias together constitute the model parameters.

Q53. Answer: B False. Too much regularisation causes underfitting — the model becomes too simple to capture real patterns. There is an optimal α value.

Q54. Answer: B The gradient points in the direction of steepest **ascent**. We move in the **opposite** direction (subtract the gradient) to descend toward the minimum.

Q55. Answer: B W has shape [K × f] — K rows (one per class), each with f weights (one per feature).

Q56. Answer: B Correct order: Compute output (ŷ) → Compute loss → Compute gradient → Update weights. This is the standard forward pass + backward pass training loop.

Q57. Answer: B $f(x^{(i)}; \theta)$ computes $\hat{y}$ — the model output for input $x^{(i)}$ given parameters $\theta = \{w, b\}$.

Q58. Answer: B The slides state: "The sigmoid function has other desirable properties such as its slope being steepest around the decision boundary."

Q59. Answer: C L1 prefers sparse solutions — it drives many weights to exactly zero while allowing others to remain relatively large.

Q60. Answer: B This is a classic overfitting example — 100% training accuracy but failure on unseen data because the model memorised the training set rather than learning general patterns.

WEEK 2 — Feed Forward Neural Networks

---

1. Important Topics

FFNs vs Logistic Regression

Activation functions (sigmoid, tanh, ReLU)

Vanishing gradient problem

Notational conventions for FFNs

Pooling (mean, sum, element-wise max)

Pretraining & embeddings

Language modelling with FFNs

Forward inference / decoding

Loss functions for FFNs

Backpropagation & computation graphs

Training details (dropout, weight init, Adam)

Training FFN language models (learning embeddings)

---

2. Definitions

Term	Definition
FFN	Fully connected feed-forward network; multiple logistic regressors joined together with hidden layers
Universal function approximator	A network with even one hidden layer can learn any function
Representation learning	Learning re-representations of inputs that cluster similar items together; useful for the task
Activation function	Non-linear function applied to weighted sum at each node (tanh, ReLU, sigmoid)
tanh	Variant of sigmoid ranging from −1 to +1; almost always better than sigmoid as hidden activation

ReLU	Rectified Linear Unit; output = z if z > 0, else 0; avoids vanishing gradient
Vanishing gradient	Problem where gradients become near-zero in deep networks, stopping learning; caused by saturated sigmoid/tanh
Pooling	Method to combine variable-length word embeddings into a fixed-size input (sum, mean, element-wise max)
Pretraining	Using embeddings (e.g. word2vec, GloVe) learned separately as input representations
Forward inference / decoding	Running a forward pass through the network to produce a probability distribution over outputs
Backpropagation	Algorithm to compute gradients for earlier layers by passing error signal backward through the computation graph
Computation graph	A graph where each node represents an operation in computing a mathematical expression
Chain rule	Calculus rule for derivative of composite functions; foundation of backpropagation
Dropout	Regularisation technique: randomly block some units from participating in training
Adam optimiser	Gradient descent variant that adapts learning rate per parameter based on past gradients
Embedding matrix E	Matrix of shape [d × |V|]; each column is a word embedding vector of dimension d

One-hot vector	Vector of length |V| with 1 at word index and 0 everywhere else; used to look up embeddings
Freezing embeddings	Keeping embedding matrix E fixed during training (not updated)
Language model	Predicts the next word given N−1 previous words; outputs probability distribution over vocabulary

---

3. Key Concepts

FFN vs Logistic Regression:

LR = one layer; FFN = multiple layers with hidden layers

LR can only solve linearly separable problems; FFNs can learn any function

FFNs avoid manual feature engineering; features are learned automatically

Hidden layers require backpropagation (not simple gradient descent as in LR)

Activation Functions:

Sigmoid/tanh saturate at high values → derivatives near 0 → vanishing gradient

ReLU derivative is 1 for $z > 0$ → no vanishing gradient problem

tanh preferred over sigmoid for hidden layers (better range: −1 to +1)

Softmax used at output layer for classification

Notation (key):

Superscripts in square brackets = layer index: a[0] = input, a[1] = hidden, etc.

z[i] = pre-activation value = W[i]a[i−1] + b[i]

a[i] = g(z[i]) where g is the activation function

Without non-linear activations, any deep network collapses to a single linear layer

Pooling:

Needed when combining word embeddings of variable-length inputs

Mean pooling: average all word embedding vectors

Sum pooling: sum all word embedding vectors

Element-wise max also possible

Language Modelling with FFN:

Takes N−1 previous words, predicts next word

Each word looked up in embedding matrix E via one-hot vector multiplication

Embeddings concatenated → hidden layer → softmax output over vocabulary

Can freeze E (use pretrained) or learn E during training

Computation Graph & Backprop:

Break expression into individual operations → graph of nodes

Forward pass: compute output left to right

Backward pass: compute gradients right to left using chain rule

Each node: upstream gradient × local gradient = downstream gradient

Backprop updates weights in ALL layers, not just the last one

Training Details:

Weights initialised to small random numbers (NOT zeros like LR)

Inputs standardised to mean=0, unit variance

Dropout randomly disables units during training to prevent overfitting

Adam adapts learning rate per parameter; handles noisy/sparse gradients better

Non-convex optimisation (unlike LR which is convex) — no guarantee of global optimum

---

4. Key Equations

Forward pass (general n-layer FFN):

$z[i] = W[i]a[i-1] + b[i]$

$a[i] = g(z[i])$

Why non-linearity matters:

With no activation: $W2 = (W[2]W[1])x$ → just one linear layer

Non-linear activations give the network real depth/power

Sigmoid derivative:

$d\sigma/dz = \sigma(z)(1 - \sigma(z))$

ReLU derivative:

dReLU/dz = 1 if z > 0, else 0

Mean pooling:

$e = (1/n) \Sigma_i e(w_i)$

CE loss for LM:

L = −log ŷ_{wt} (negative log probability of the correct next word)

Embedding lookup:

$e_{w_i} = E \times x_{one\text{-}hot}$

---

5. Things to Remember

LR can only learn **linearly separable** problems; FFNs can learn **any** function

ReLU avoids vanishing gradient; sigmoid/tanh do not

Without non-linear activation functions, deep networks = single linear layer

Weights initialised to **small random numbers** in FFNs (not zeros)

Backpropagation = backward differentiation on computation graphs

Computation graph backward pass uses **chain rule** at every node

Dropout introduced by Hinton et al. 2012 / Srivastava et al. 2014

Adam optimiser: adapts learning rate **per parameter** based on past gradients

FFN language model: input = N−1 words, output = probability over **entire vocabulary**

Embedding matrix E shape = **[d × |V|]** (d = embedding dimension, |V| = vocab size)

Embedding lookup = multiply E by one-hot vector → selects the column = word embedding

Embeddings can be **frozen** (pretrained) or **learned** during training

FFN optimisation is **non-convex** → no global optimum guarantee (unlike LR)

Pooling is needed when combining **variable-length** inputs into fixed-size FFN input

The first FFNLM: **Bengio et al., 2003**

WEEK 2 — MCQs

Section A — Questions

Q1. What is the key limitation of logistic regression that feed-forward neural networks overcome?

A) LR cannot handle numerical inputs B) LR can only learn linearly separable problems C) LR requires too much training data D) LR cannot use gradient descent

---

Q2. A neural network with a single hidden layer is described as:

A) Unable to learn complex functions B) A universal function approximator C) Limited to binary classification D) Equivalent to Naive Bayes

---

Q3. Which of the following is NOT a common activation function used in hidden layers of FFNs?

A) tanh B) ReLU C) Softmax D) Sigmoid

Q4. The tanh activation function ranges from:

A) 0 to 1 B) $-\infty$ to $+\infty$ C) -1 to $+1$ D) 0 to $+\infty$

Q5. ReLU outputs:

A) z when z > 0, else 0 B) $1/(1+e^{-z})$ C) z for all values of z D) 0 when z > 0, else z

---

Q6. What causes the vanishing gradient problem?

A) Learning rate being too high B) Too many training examples C) Sigmoid/tanh derivatives becoming near-zero at saturated values D) Using ReLU activation

---

Q7. Why does ReLU not suffer from the vanishing gradient problem?

A) Its output is always positive B) Its derivative is 1 for z > 0, not near zero C) It uses a logarithmic scale D) It applies softmax internally

---

Q8. Which activation function is used at the OUTPUT layer for multi-class classification?

A) ReLU B) tanh C) Sigmoid D) Softmax

Q9. In the FFN notation, what does z[i] represent?

A) The output of the activation function at layer i B) The pre-activation value at layer i (= W[i]a[i−1] + b[i]) C) The loss at layer i D) The gradient at layer i

---

Q10. In FFN notation, a[0] refers to:

A) The first hidden layer output B) The bias vector C) The input vector x D) The output layer

---

Q11. What happens to a deep network if no non-linear activation functions are used?

A) It becomes more powerful B) It becomes equivalent to a single linear layer C) Gradients vanish instantly D) It cannot perform forward pass

---

Q12. What is representation learning in the context of FFNs?

A) Memorising training examples B) Learning to re-represent inputs so similar items cluster together usefully for the task C) Manually designing features from domain knowledge D) Reducing the number of parameters

---

Q13. The XOR problem is used in the slides to demonstrate:

A) Why sigmoid is better than ReLU B) How pooling works C) How hidden layers enable representation learning to solve non-linearly separable problems D) Why Naive Bayes fails on large datasets

---

Q14. What is pooling used for in FFN-based NLP classification?

A) To increase vocabulary size B) To combine variable-length word embeddings into a single fixed-size input C) To regularise the network D) To compute gradients faster

---

Q15. Which of the following is an example of a pooling method mentioned in the slides?

A) Max-norm pooling B) Dropout pooling C) Mean pooling (average of all word embeddings) D) Softmax pooling

---

Q16. Pretraining in the context of FFNs refers to:

A) Training the network twice on the same data B) Using embeddings (like word2vec or GloVe) learned separately as input representations C) Initialising weights to zero before training D) Training only the output layer first

---

Q17. Word2vec and GloVe are examples of:

A) Activation functions B) Loss functions C) Pretrained static word embeddings D) Regularisation techniques

---

Q18. A Feed-Forward Neural Language Model (FFNLM) takes as input:

A) The entire sentence B) One word at a time C) N−1 previous words and predicts the next word D) N+1 future words

---

Q19. In an FFNLM, how is a word looked up in the embedding matrix?

A) By multiplying the embedding matrix by the word's frequency count B) By multiplying the embedding matrix E by a one-hot vector for that word C) By adding the word index to E D) By applying softmax to E

Q20. The embedding matrix E has shape:

A) [|V| × |V|] B) [d × |V|] where d = embedding dimension, |V| = vocabulary size C) [|V| × d] with rows as embeddings D) [N × d] where N = context window size

---

Q21. In forward inference of an FFNLM, the output layer produces:

A) The raw embedding of the next word B) A probability distribution over all words in the vocabulary C) A single scalar confidence score D) The hidden layer activations

---

Q22. What does "freezing" the embedding matrix mean during training?

A) Using dropout on the embedding layer B) Keeping E fixed at pretrained values and not updating it during training C) Reducing the dimensionality of E D) Applying L2 regularisation to E

Q23. When embeddings are learned during FFN training (not frozen), the embedding matrix E is treated as:

A) A fixed input B) Another weight matrix updated by backpropagation C) A bias term D) A regularisation parameter

---

Q24. The cross-entropy loss for language modelling is:

A) $-\Sigma \square\, y \square \log \hat{y} \square$ summed over all classes B) $-\log \hat{y}_{wt}$ where wt is the correct next word C) Mean squared error between $\hat{y}$ and y D) The sigmoid of the output

---

Q25. What is a computation graph?

A) A graph showing training accuracy over time B) A representation of computing a mathematical expression, broken into individual operations as nodes C) A visualisation of the weight matrix D) A graph of the vocabulary distribution

---

Q26. In backward differentiation on a computation graph, gradients flow:

A) From input nodes to output nodes B) Randomly through the graph C) From the output node back to earlier nodes D) Only through bias nodes

---

Q27. At each node during backpropagation, the downstream gradient is computed as:

A) Upstream gradient + local gradient B) Upstream gradient × local gradient C) Local gradient / upstream gradient D) Upstream gradient − local gradient

---

Q28. The chain rule states that for $f(x) = u(v(x))$, the derivative df/dx is:

A) $du/dx + dv/dx$ B) $(du/dv) \times (dv/dx)$ C) du/dv only D) dv/dx only

---

Q29. Backpropagation is needed in FFNs (compared to LR) because:

A) FFNs use a different loss function B) Gradients for earlier layers cannot be directly computed without passing error backward through layers C) FFNs have no bias terms D) LR does not use gradient descent

---

Q30. The derivative of the sigmoid function $\sigma(z)$ is:

A) $\sigma(z) + (1 - \sigma(z))$ B) $\sigma(z)(1 - \sigma(z))$ C) $1 - \sigma(z)^2$ D) $\sigma(z)^2$

Q31. The derivative of ReLU is:

A) $\sigma(z)(1-\sigma(z))$ B) Always 1 C) 1 if $z > 0$, else 0 D) z if $z > 0$, else 1

Q32. Why are FFN weights initialised to small random numbers rather than zeros?

A) Zero initialisation causes overfitting B) Zeros cause all units to learn identical features and prevents learning C) Random initialisation is required by the softmax function D) Zero weights cause the loss to be undefined

---

Q33. What does dropout do during training?

A) Removes low-weight features permanently B) Randomly blocks some units and their connections from participating in training C) Reduces the learning rate gradually D) Adds noise to the loss function

---

Q34. Dropout was introduced by:

A) Bengio et al., 2003 B) Rumelhart et al., 1986 C) Hinton et al. 2012 / Srivastava et al. 2014 D) Mikolov et al., 2013

Q35. How does Adam optimiser differ from standard gradient descent?

A) Adam uses second-order derivatives B) Adam adapts the learning rate per parameter based on past gradients C) Adam removes the need for a loss function D) Adam uses batch size of 1 always

---

Q36. FFN optimisation is described as:

A) Convex — guaranteed to find global optimum B) Non-convex — no guarantee of global optimum C) Linear — solved analytically D) Identical to LR optimisation

---

Q37. Which framework is mentioned in the slides for implementing computation graphs on GPUs?

A) TensorFlow and Keras B) PyTorch and TensorFlow C) Scikit-learn and NumPy D) Theano and Caffe

---

Q38. In the FFN notation with shape variables, $W \in \mathbb{R}^{(n1 \times n0)}$ represents:

A) The output weight matrix B) The weight matrix connecting input layer (n0 units) to hidden layer (n1 units) C) The bias vector D) The embedding matrix

---

Q39. What is the shape of the output matrix $\hat{Y}$ when processing m examples with d_o output classes?

A) $[d_o \times m]$ B) $[m \times d_o]$ C) $[m \times d]$ D) $[d_o \times d_o]$

Q40. The first Feed-Forward Neural Language Model was introduced by:

A) Rumelhart et al., 1986 B) Mikolov et al., 2013 C) Bengio et al., 2003 D) Hinton et al., 2012

Q41. What is the advantage of NLMs over N-gram models? (Select the most complete answer)

A) NLMs are faster and more interpretable B) NLMs handle longer histories, generalise better, and predict words more accurately C) NLMs require less data to train D) NLMs don't need embeddings

---

Q42. What is the downside of NLMs compared to N-gram models?

A) NLMs cannot model language B) NLMs are more complex, slower, less energy-efficient, and less interpretable C) NLMs cannot use gradient descent D) NLMs require hand-crafted features

---

Q43. In an FFNLM with a 3-word context window, how many one-hot vectors are created?

A) 1 B) 2 C) 3 D) Equal to vocabulary size

Q44. When training the FFNLM end-to-end (including embeddings), which parameters are in θ?

A) W and U only B) E, W, U, and b C) E and b only D) W, U, b (E is always frozen)

Q45. Why is a single embedding matrix E shared across all context positions in the FFNLM?

A) It reduces the vocabulary size B) Each word should have one vector regardless of which context position it appears in C) Using separate matrices causes vanishing gradients D) Shared matrices require less computation

---

Q46. Training an FFNLM results in:

A) Only a language model (word predictor) B) Only a set of word embeddings C) Both a language model AND word embeddings that can be used for other tasks D) A pretrained softmax classifier

---

Q47. In computing L(a,b,c) = c(a+2b) with a=3, b=1, c=−2, what is the output L?

A) 5 B) −10 C) 10 D) −5

Q48. In the computation graph for L(a,b,c) = c(a+2b), what is $\partial L/\partial c$?

A) −2 B) 5 C) 2 D) −10

Q49. Inputs to FFNs are standardised to have:

A) Maximum value of 1 B) Mean = 0 and unit variance C) Mean = 0.5 and variance = 0.5 D) Values between 0 and 1

Q50. TRUE or FALSE: In an FFN, the LR-style gradient update $(\hat{y}-y)x$□ correctly updates ALL layers.

A) True — it works for all layers B) False — it only gives correct updates for the last (output) layer C) True — backprop is just repeated application of this formula D) False — it only works for the first (input) layer

---

Q51. Which of the following best describes the role of hidden layers in FFNs?

A) They apply softmax to the inputs B) They learn intermediate representations that make the final classification easier C) They store the training data D) They compute the loss function

---

Q52. Contextual embeddings (from deep neural network hidden layers) are described as:

A) Identical to word2vec embeddings B) Static embeddings that don't change with context C) Much more powerful than simple static word embeddings like word2vec D) Less useful than one-hot vectors

---

Q53. What does the notation '[...; ...]' indicate in the FFNLM equations?

A) Matrix multiplication B) Concatenation of vectors C) Element-wise addition D) Dot product

---

Q54. In a 2-layer FFN for 3-way sentiment classification, the final output layer uses which activation?

A) ReLU B) Sigmoid C) tanh D) Softmax (for 3-way classification)

Q55. Which statement about tanh compared to sigmoid is correct per the slides?

A) tanh has the same range as sigmoid B) tanh is a variant of sigmoid ranging from −1 to +1 and is almost always better C) tanh causes more vanishing gradient problems than sigmoid D) tanh cannot be used in hidden layers

Section B — Answers & Explanations

Q1. B — LR can only solve linearly separable problems. FFNs with hidden layers can learn any function.

Q2. B — Directly from the slides: a network with even one hidden layer is a universal function approximator.

Q3. C — Softmax is used at the OUTPUT layer, not hidden layers. tanh, ReLU, and sigmoid are used as hidden layer activations.

Q4. C — tanh ranges from −1 to +1. Sigmoid ranges from 0 to 1.

Q5. A — ReLU(z) = z if z > 0, else 0. It is linear for positive values and zero for negative.

Q6. C — Sigmoid/tanh saturate at extreme values, producing derivatives very close to 0, which shrinks the gradient signal to nothing during backprop.

Q7. B — ReLU's derivative is 1 for z > 0, so the gradient signal doesn't shrink when passing through ReLU nodes.

Q8. D — Softmax is used at the output for multi-class classification; it produces a probability distribution over K classes.

Q9. B — z[i] is the pre-activation value: z[i] = W[i]a[i−1] + b[i]. The activation function is applied after to get a[i].

Q10. C — a[0] is the generalised notation for the input vector x (layer 0 = input layer).

Q11. B — Without non-linearities, W2 = (W[2]W[1])x — just a single linear transformation, no matter how many layers.

Q12. B — Representation learning = learning to re-represent inputs so similar items cluster together in a way that helps solve the task.

Q13. C — XOR is not linearly separable in original space, but hidden layers learn a new representation where it becomes solvable.

Q14. B — Pooling aggregates multiple word embeddings (variable-length) into a single fixed-size vector for FFN input.

Q15. C — Mean pooling (average of all word embeddings) is explicitly mentioned. Also sum and element-wise max.

Q16. B — Pretraining = using embeddings from a separately trained algorithm (word2vec, GloVe) as the initial input representations.

Q17. C — word2vec and GloVe are pretrained static word embeddings mentioned as examples in the slides.

Q18. C — An FFNLM takes N−1 previous words as input and outputs a probability over the next word.

Q19. B — Embedding lookup = E × one-hot vector. The one-hot selects the column of E corresponding to that word.

Q20. B — E has shape [d × |V|]: d rows (embedding dimensions), |V| columns (one per vocabulary word).

Q21. B — The output layer applies softmax to produce a probability distribution over all words in the vocabulary.

Q22. B — Freezing = holding E constant at pretrained values, only updating W, U, b during training.

Q23. B — When learning embeddings during training, E is treated as another weight matrix and updated via backpropagation.

Q24. B — For LM, the loss simplifies to −log ŷ_{wt}: the negative log probability of the correct next word.

Q25. B — A computation graph breaks a mathematical expression into individual operations (nodes) and shows how they connect.

Q26. C — Backward differentiation passes gradients from the output node back toward the input nodes.

Q27. B — At each node: downstream gradient = upstream gradient × local gradient (chain rule).

Q28. B — Chain rule: df/dx = (du/dv)(dv/dx). The derivative of a composite function is the product of the outer and inner derivatives.

Q29. B — In deep networks, you can't directly compute gradients for early layers — error must be propagated backward layer by layer.

Q30. B — $d\sigma/dz = \sigma(z)(1 - \sigma(z))$. This elegant form means the derivative depends only on the sigmoid output itself.

Q31. C — ReLU derivative: 1 if $z > 0$, 0 otherwise. This is why it avoids vanishing gradients (derivative is never near-zero for positive inputs).

Q32. B — If all weights are zero, all units compute the same value and gradients are identical — no differentiation occurs and learning fails.

Q33. B — Dropout randomly disables units and their connections during training, preventing co-adaptation and reducing overfitting.

Q34. C — Dropout was introduced by Hinton et al. 2012 and Srivastava et al. 2014, as explicitly cited in the slides.

Q35. B — Adam adapts the learning rate for each parameter individually based on past gradient history, handling sparse/noisy gradients better.

Q36. B — FFN optimisation is non-convex (unlike LR which is convex), so there is no guarantee of finding the global optimum.

Q37. B — The slides explicitly mention PyTorch (Paszke et al., 2017) and TensorFlow (Abadi et al., 2015).

Q38. B — $W \in \mathbb{R}^{(n1 \times n0)}$ connects n0 input units to n1 hidden units. Rows = destination layer size, columns = source layer size.

Q39. B — Output matrix $\hat{Y}$ has shape $[m \times d_o]$: m rows (one per input example), d_o columns (one per output class).

Q40. C — Bengio et al., 2003 introduced the first FFNLM, explicitly cited in the slides.

Q41. B — NLMs handle longer histories, generalise better over similar contexts, and predict words more accurately than N-gram models.

Q42. B — NLMs are more complex, slower and less energy-efficient to train, and less interpretable than N-gram models.

Q43. C — A 3-word context window → 3 one-hot vectors, one for each of the 3 preceding words.

Q44. B — When learning embeddings during training, $\theta = \{E, W, U, b\}$ — all four sets of parameters.

Q45. B — A single shared E ensures each word has one consistent embedding vector regardless of which position it appears in the context window.

Q46. C — Training an FFNLM produces both a language model (word predictor) AND useful word embeddings for other tasks.

Q47. B — $d=2b=2$, $e=a+d=5$, $L=c\times e=-2\times 5=-10$.

Q48. B — $\partial L/\partial c = \partial(c \times e)/\partial c = e = a+2b = 3+2 = 5$.

Q49. B — Inputs are standardised to mean=0 and unit variance, as stated in the training details.

Q50. B — False. The LR gradient formula $(\hat{y}-y)x$□ only correctly updates the last layer. Earlier layers need backpropagation to get correct gradients.

Q51. B — Hidden layers learn intermediate representations that transform the input into a form that makes the final output layer's job easier.

Q52. C — The slides describe contextual embeddings as "much more powerful" than simple static embeddings like word2vec.

Q53. B — '[...; ...]' notation indicates concatenation of vectors, as noted in the FFNLM decoding equations.

Q54. D — 3-way (3-class) classification uses softmax at the output layer, not sigmoid (which is for binary).

Q55. B — Directly from the slides: tanh is "a variant of the sigmoid that ranges from −1 to +1" and "almost always better."

Week 3

WEEK 3 — PyTorch and Hugging Face

---

1. Important Topics

Matrix Multiplication (dimensions & rules)

PyTorch (overview, key features)

Hugging Face (overview, key libraries)

---

2. Definitions

Term	Definition
PyTorch	Open-source end-to-end ML framework for fast, flexible AI experimentation and production
torch.Tensor	PyTorch's core class for homogeneous multidimensional arrays; like NumPy arrays but GPU-capable
Automatic differentiation	PyTorch's tape-based system for computing gradients automatically
TorchScript	Tool to export PyTorch models from Python to production environments where Python is disadvantageous
TorchServe	Tool for serving PyTorch models in production (multi-model serving, logging, REST endpoints)
Hugging Face (HF)	Platform/company known for its Transformers library and the HF Hub for sharing ML models and datasets
HF Transformers library	Python package with open-source implementations of transformer models for text, image, audio; compatible with PyTorch, TensorFlow, and JAX
HF Hub	Platform for hosting Git-based repositories, models, datasets, and web applications

BLOOM	Open large multilingual language model developed partly by HF
Gradio	HF library for creating ML demos
CUDA	NVIDIA's GPU computing platform; PyTorch tensors can run directly on CUDA-capable GPUs

---

3. Key Concepts

Matrix Multiplication Rules:

Matrix A [m × x] multiplied by Matrix B [x × n] → Result [m × n]

Inner dimensions must match (x = x)

Outer dimensions give result shape (m × n)

Tip: rows in result = rows of first matrix; columns in result = columns of second matrix

PyTorch Key Facts:

Started in 2017 as a Python port of Torch (originally Lua-based, created by Idiap Research Institute at EPFL)

Since Sep 2022 governed by the **PyTorch Foundation** (subsidiary of the Linux Foundation)

Licensed under **modified BSD license** (free and open-source)

De facto standard ML framework for academic research and commercial development

Two high-level features: (1) Tensor computing with GPU acceleration, (2) Deep NNs built on tape-based automatic differentiation

Supports distributed training via asynchronous execution

Experimental support for iOS and Android deployment

Hugging Face Key Facts:

Founded in **2016** originally as a chatbot app for teenagers; pivoted to ML platform

Originally named library: **pytorch-pretrained-bert** (now called Transformers)

HF offers: Transformers, Datasets, Evaluate, Simulate, Gradio libraries

Partners with AWS (HF products available within AWS; BLOOM runs on AWS's Trainium chip)

---

4. Things to Remember

Matrix multiplication: **(m×x)(x×n) = (m×n)** — inner dimensions must match

PyTorch tensors can run on CPU **or** CUDA-capable NVIDIA GPU (unlike NumPy arrays which are CPU only)

PyTorch founded **2017**, governed by PyTorch Foundation since **Sep 2022**

HF founded **2016**; transformers library originally called **pytorch-pretrained-bert**

HF Transformers compatible with **PyTorch, TensorFlow, and JAX**

BLOOM = HF's open multilingual LLM

TorchScript = for production deployment; TorchServe = for serving models

Week 3 lab submissions: (1) FFN reimplemented in PyTorch, (2) HF coding task

---

WEEK 3 — MCQs

Section A — Questions

Q1. For matrix multiplication A × B to be valid, which condition must hold?

A) Number of rows in A must equal number of rows in B B) Number of columns in A must equal number of rows in B C) Number of rows in A must equal number of columns in B D) Both matrices must be square

---

Q2. If matrix A has shape [3 × 4] and matrix B has shape [4 × 5], what is the shape of the resulting matrix?

A) [4 × 4] B) [3 × 5] C) [5 × 3] D) [3 × 4]

Q3. If matrix A has shape [2 × 3] and matrix B has shape [4 × 2], can A × B be computed?

A) Yes, result is [2 × 2] B) Yes, result is [3 × 4] C) No, the inner dimensions do not match D) Yes, result is [2 × 4]

Q4. In matrix multiplication of [m × x] by [x × n], what are the dimensions of the result?

A) [x × x] B) [m × n] C) [n × m] D) [x × m]

Q5. PyTorch was originally started in which year?

A) 2012 B) 2015 C) 2017 D) 2020

Q6. PyTorch originated as a Python port of which library?

A) TensorFlow B) Theano C) Torch (Lua-based) D) Caffe

Q7. Torch (the predecessor of PyTorch) was created by:

A) Google Brain B) Idiap Research Institute at EPFL C) Facebook AI Research D) OpenAI

Q8. As of September 2022, PyTorch is governed by:

A) Facebook/Meta B) Google C) The PyTorch Foundation, a subsidiary of the Linux Foundation D) NVIDIA

Q9. PyTorch is released under which license?

A) Apache 2.0 B) MIT License C) GPL v3 D) Modified BSD license

---

Q10. Which of the following best describes PyTorch's torch.Tensor class?

A) A class only for 1D arrays of integers B) A homogeneous multidimensional array class similar to NumPy but also GPU-capable C) A class exclusively for matrix multiplication D) A class for storing string data

---

Q11. What is the key difference between PyTorch tensors and NumPy arrays?

A) PyTorch tensors only support integer data types B) PyTorch tensors can be operated directly on a CUDA-capable NVIDIA GPU C) NumPy arrays are faster than PyTorch tensors D) PyTorch tensors cannot be used for matrix multiplication

---

Q12. PyTorch's two main high-level features are:

A) Image recognition and speech synthesis B) Tensor computing with GPU acceleration and deep NNs built on tape-based automatic differentiation C) Model serving and dataset processing D) NLP and computer vision pipelines

---

Q13. What is TorchScript used for?

A) Writing PyTorch tutorials B) Exporting PyTorch models to production environments where Python may be disadvantageous C) Automatic hyperparameter tuning D) Visualising computation graphs

---

Q14. What is TorchServe used for?

A) Training models faster B) Serving PyTorch models in production with multi-model serving, logging, metrics, and REST endpoints C) Compiling PyTorch code to C++ D) Storing datasets for training

---

Q15. PyTorch supports distributed training through:

A) Only synchronous gradient updates B) Native support for asynchronous execution of collective operations and peer-to-peer communication C) A separate distributed library not included in PyTorch D) Requiring TensorFlow as a backend

---

Q16. Hugging Face was originally founded in which year?

A) 2012 B) 2014 C) 2016 D) 2019

Q17. What was Hugging Face's original purpose before pivoting to ML?

A) Developing a search engine B) Developing a chatbot app targeted at teenagers C) Building a cloud computing platform D) Creating a dataset sharing website

---

Q18. What was the original name of the Hugging Face Transformers library?

A) huggingface-bert B) torch-nlp C) pytorch-pretrained-bert D) transformer-hub

---

Q19. The HF Transformers library is compatible with which deep learning frameworks?

A) PyTorch only B) TensorFlow only C) PyTorch, TensorFlow, and JAX D) PyTorch and Keras only

---

Q20. What is the Hugging Face Hub?

A) A hardware accelerator for training models B) A platform for hosting Git-based repositories, models, datasets, and web applications C) A Python library for data preprocessing D) A service for deploying models on mobile devices

---

Q21. Which of the following is NOT a library offered by Hugging Face?

A) Datasets B) Evaluate C) Gradio D) TorchServe

Q22. BLOOM is described in the slides as:

A) A dataset for NLP benchmarking B) An open large multilingual language model developed partly by HF C) A PyTorch optimisation library D) A tool for model evaluation

---

Q23. HF is partnering with which cloud provider to make HF products available as building blocks?

A) Google Cloud B) Microsoft Azure C) AWS (Amazon Web Services) D) IBM Cloud

---

Q24. BLOOM runs on which AWS hardware?

A) AWS Inferentia B) AWS Graviton C) AWS Trainium D) AWS Lambda

---

Q25. What is Gradio, as mentioned in the slides?

A) A HF library for dataset processing B) A HF library for creating ML demos C) A HF library for model evaluation D) A HF library for simulation

---

Q26. Which of the following correctly lists all four HF libraries mentioned in the slides?

A) Transformers, Datasets, Evaluate, Gradio B) Transformers, TorchServe, Evaluate, Simulate C) Datasets, Gradio, BLOOM, Evaluate D) Transformers, Datasets, Simulate, TorchScript

---

Q27. For the Week 3 lab, Task 2 of the PyTorch lab requires:

A) Working through the 8 PyTorch tutorial sections B) Reimplementing the feed-forward neural network from Week 2 in PyTorch C) Training a HuggingFace model on AGnews D) Writing a matrix multiplication function from scratch

---

Q28. The matrix multiplication lab requires students to:

A) Use numpy.matmul as the primary method B) Write a Python function from scratch without using existing array functions, and compare results with numpy.matmul C) Use PyTorch tensors to multiply matrices D) Implement backpropagation for matrix multiplication

---

Q29. If matrix A is [5 × 2] and matrix B is [2 × 1], what is the shape of A × B?

A) [2 × 2] B) [5 × 1] C) [1 × 5] D) [5 × 2]

Q30. PyTorch is described in the slides as:

A) A niche research tool not widely adopted B) The de facto standard ML framework for academic research and much commercial development C) Primarily used for computer vision only D) A framework that requires CUDA to function

---

Section B — Answers & Explanations

Q1. B — For A×B to be valid, the number of columns in A must equal the number of rows in B (inner dimensions must match).

Q2. B — [3×4] × [4×5]: inner dimensions (4=4) match; result is [3×5] (outer dimensions).

Q3. C — A is [2×3], B is [4×2]. For A×B, columns of A (3) must equal rows of B (4). 3≠4, so not compatible.

Q4. B — [m×x] × [x×n] = [m×n]. Inner dimensions x cancel; outer dimensions m and n give the result shape.

Q5. C — PyTorch started in 2017 as explicitly stated in the slides.

Q6. C — PyTorch is a Python port of Torch, which was Lua-based.

Q7. B — Torch was created by the Idiap Research Institute at EPFL, as stated in the slides.

Q8. C — Since September 2022, PyTorch is governed by the PyTorch Foundation, a subsidiary of the Linux Foundation.

Q9. D — PyTorch is released under the modified BSD license, as explicitly stated.

Q10. B — torch.Tensor is for homogeneous multidimensional arrays, similar to NumPy but also GPU-capable.

Q11. B — The key difference is that PyTorch tensors can run directly on CUDA-capable NVIDIA GPUs; NumPy arrays cannot.

Q12. B — The two high-level features listed are: (1) Tensor computing with GPU acceleration, (2) Deep NNs on tape-based automatic differentiation.

Q13. B — TorchScript is for exporting trained PyTorch models to production environments where Python may be disadvantageous.

Q14. B — TorchServe handles production serving with multi-model serving, logging, metrics, and RESTful endpoints.

Q15. B — PyTorch supports distributed training via native support for asynchronous execution of collective operations and peer-to-peer communication.

Q16. C — Hugging Face was founded in 2016.

Q17. B — HF originally developed a chatbot app targeted at teenagers before pivoting to become an ML platform.

Q18. C — The original name was pytorch-pretrained-bert, as stated in the slides.

Q19. C — HF Transformers is compatible with PyTorch, TensorFlow, and JAX — all three explicitly mentioned.

Q20. B — The HF Hub is a platform for hosting Git-based repositories, models, datasets, and web applications.

Q21. D — TorchServe is a PyTorch tool, not a Hugging Face library. HF libraries are Transformers, Datasets, Evaluate, Simulate, and Gradio.

Q22. B — BLOOM is described as an open large multilingual language model developed partly by HF.

Q23. C — HF is partnering with AWS, whose customers can use HF products within AWS as building blocks.

Q24. C — BLOOM runs on AWS's proprietary ML chip called Trainium, as stated in the slides.

Q25. B — Gradio is HF's library for creating machine learning demos.

Q26. A — The four HF libraries explicitly listed are: Transformers, Datasets, Evaluate, and Gradio (Simulate is also mentioned but Gradio replaces TorchServe/TorchScript which are PyTorch tools).

Q27. B — PyTorch lab Task 2 requires reimplementing the Week 2 feed-forward neural network in PyTorch using the AGnews dataset.

Q28. B — The lab requires a from-scratch Python implementation (no existing array functions), with results compared against numpy.matmul.

Q29. B — [5×2] × [2×1]: inner dimensions (2=2) match; result is [5×1].

Q30. B — The slides explicitly state PyTorch "has become the de facto standard ML framework for academic research as well as a lot of commercial development work."

Week 4

WEEK 4 — Recurrent Neural Networks

---

1. Important Topics

Why RNNs (limitations of LR/FFN for sequences)

RNN architecture & copy-back mechanism

Inference in RNNs (forward pass, weight sharing)

Backpropagation Through Time (BPTT)

RNNs for Language Modelling

Teacher Forcing

Weight Tying

RNNs for Sequence Labelling

Autoregressive Text Generation

RNNs for Sequence Classification

End-to-end learning

---

2. Definitions

Term	Definition
RNN	2-layer feedforward network with a copy-back mechanism; processes sequences by maintaining a hidden state across time steps
Copy-back mechanism	Copies hidden state from previous time step and concatenates it with current input
Hidden state (h)	Internal memory of the RNN; embeds traces of all previous inputs
BPTT	Backpropagation Through Time; backprop applied to the unrolled RNN computation graph
Teacher forcing	Always feeding the correct previous word (not the model's prediction) as input during training
Weight tying	Using the same weight matrix for both the input embedding E and output matrix V (with transpose); saves parameters
Autoregressive generation	Incrementally generating words by sampling each next word conditioned on previous choices
Sequence labelling	Producing a label for each input token (e.g. POS tagging)

Sequence classification	Classifying the whole sequence using the final hidden state (e.g. sentiment analysis)
End-to-end learning	Loss from a downstream task propagated back through all weights in the entire model
Weight sharing	U, V, W matrices are the same across all time steps; only x, h, y change per step
Perplexity	Measure of LM quality; weight tying results in lower perplexity

---

3. Key Concepts

Why RNNs?

LR and FFNs have fixed-length inputs — cannot model long-range/discontinuous dependencies

Sliding window FFNs still miss dependencies outside the window

RNNs extend prior context to any length via the hidden state

RNN Architecture:

First introduced by Elman (1990)

Hidden state h_t depends on current input x_t AND previous hidden state h_{t-1}

Weights U (input→hidden), W (hidden→hidden), V (hidden→output) are **shared across all time steps**

x_t, h_t, y_t change at every step; U, V, W do not

Matrix Dimensions:

Input→Hidden weight matrix U: shape [dh × din]

Hidden→Hidden weight matrix W: shape [dh × dh]

Hidden→Output weight matrix V: shape [dout × dh]

Forward Pass equations (per time step t):

$h_t = g(U \cdot x_t + W \cdot h_{t-1} + b)$ → g is typically tanh

$y_t = f(V \cdot h_t + c)$ → f is typically softmax

Total loss L = sum of losses L_t at each time step

Complexity:

Forward and backward pass both take time proportional to τ (number of time steps)

Cannot be parallelised — each step depends on the previous step

Training (BPTT):

Unroll the RNN over all n time steps → treat as one big computation graph

Forward pass: compute and save all h_t and y_t

Backward pass: propagate gradients backward through the unrolled graph

All hidden states except the last receive gradients from **two sources**: from $\hat{y}_t$ AND from h_{t+1}

For very long sequences: unroll into fixed-length segments (not feasible to unroll entire sequence)

RNN Language Model:

Words encoded as one-hot vectors → looked up in embedding matrix E

Embedding concatenated with previous hidden state

Output: softmax over entire vocabulary

No fixed-context problem (unlike FFN LMs) — hidden state can represent all preceding words

Training uses self-supervision: predict next word at every step

Teacher Forcing:

During training, always use the correct previous word as input (not the model's prediction)

Avoids compounding errors during training

Weight Tying:

V maps hidden→logits (one per vocab item); E maps one-hot→hidden

They learn effectively the same mapping in opposite directions

If shape of V = [|V| × dh] and E = [dh × |V|]: set $V = E^T$

Benefits: fewer parameters, faster training, lower perplexity

Sequence Labelling:

Output produced at every time step

Input = pretrained word embeddings; output = softmax over tag set

Loss = CE at every step

Autoregressive Generation:

Start with <s> token

Sample next word from softmax output

Feed sampled word's embedding as next input

Continue until </s> is sampled or length limit reached

Applications: machine translation, summarisation, QA (primed with appropriate context)

Sequence Classification:

Only the FINAL hidden state $h_\square$ is used to represent the whole sequence

Passed to a classifier (typically a 2-layer FFN)

No intermediate outputs → loss only from final classification

Alternative: pool all hidden states (element-wise mean or max) instead of using only the last

Vanishing Gradient in RNNs:

Simple RNNs fail on long inputs due to vanishing gradients

Modern solution: gated architectures like **LSTMs** (explicitly decide what to remember/forget)

---

4. Key Equations

Forward pass:

$h_\square = \tanh(U \cdot x_\square + W \cdot h_{\square-1} + b)$

$o_\square = V \cdot h_\square + c$

$\hat{y}_\square = \text{softmax}(o_\square)$

Total loss:

$L = \Sigma_\square\ L(t) = \Sigma_\square\ -\log p(y(t) \mid x(1),...,x(t))$

CE loss at time t (LM):

$L(t) = -\log \hat{y}_{\square}[w_{\square}]$ (negative log prob of correct next word)

Probability of whole sequence:

$P(w_1,...,w_{\square}) = \Pi\ P(w_i \mid w_1,...,w_{i-1})$

Gradient at output (time t):

$\nabla_{o(t)}\ L = \hat{y}(t) - y(t)$ (one-hot gold label)

Gradient at h(τ) (final step):

$\nabla_{h(\tau)}\ L = V^T \cdot \nabla_{o(\tau)}\ L$

Gradient at h(t) for t < τ (two sources):

$\nabla_{h(t)}\ L = V^T \cdot \nabla_{o(t)} L + W^T \cdot \nabla_{h(t+1)} L$

---

5. Things to Remember

RNNs introduced by **Elman, 1990**

RNN LMs introduced by **Mikolov et al., 2010**

U, V, W are **shared** across time steps; x, h, y change each step

Hidden states have **two gradient sources** (except the last): from output AND from next hidden state

BPTT cannot be parallelised — sequential dependency

Teacher forcing: always use **correct** previous word, not model's prediction

Weight tying: $V = E^T \rightarrow$ fewer parameters, lower perplexity

Sequence classification uses **final hidden state** (or pooled hidden states); no intermediate loss

Sequence labelling produces output at **every** time step

Autoregressive generation primed with task-specific context (not just <s>) for MT, summarisation etc.

Simple RNNs have **short-term memory** — traces of earlier inputs fade

Solution to vanishing gradient in RNNs = **LSTMs** (gated architectures)

End-to-end learning: loss propagated through the classifier AND back through all RNN weights

---

WEEK 4 — MCQs

Section A — Questions

Q1. Why are standard FFNs insufficient for modelling language sequences with long-range dependencies?

A) FFNs cannot use gradient descent B) FFNs have fixed-length inputs and process inputs entirely separately C) FFNs require too much memory D) FFNs cannot use softmax

---

Q2. RNNs were first introduced by:

A) Mikolov et al., 2010 B) Bengio et al., 2003 C) Elman, 1990 D) Rumelhart et al., 1986

Q3. The copy-back mechanism in RNNs:

A) Copies the output layer back to the input B) Copies the hidden state from the previous time step and concatenates it to the current input C) Copies the weight matrices at each time step D) Copies the loss value back to the hidden layer

---

Q4. Which weight matrices are SHARED across all time steps in an RNN?

A) x□, h□, y□ B) Only V C) U, V, and W D) Only U and W

Q5. What changes at every time step in an RNN?

A) Weight matrices U, V, W B) The bias vectors only C) x□, h□, and y□ D) The learning rate

---

Q6. Given input dimension din, hidden dimension dh, and output dimension dout, what is the shape of weight matrix W (hidden-to-hidden)?

A) [din × dh] B) [dh × din] C) [dh × dh] D) [dout × dh]

Q7. What is the shape of the input-to-hidden weight matrix U?

A) [dh × din] B) [din × dout] C) [dh × dh] D) [dout × din]

Q8. The typical activation function g used for computing h□ in an RNN is:

A) ReLU B) Softmax C) tanh D) Sigmoid

Q9. The typical activation function f used for computing output y□ in an RNN is:

A) tanh B) ReLU C) Sigmoid D) Softmax

Q10. The total loss L across a sequence is:

A) The loss at only the final time step B) The average loss across all time steps C) The sum of losses at all time steps D) The maximum loss across all time steps

---

Q11. Why cannot RNN forward and backward passes be parallelised?

A) Because GPU memory is insufficient B) Because each time step must be fully computed before the next step can begin C) Because weight matrices are different at each step D) Because softmax requires sequential computation

---

Q12. Backpropagation Through Time (BPTT) refers to:

A) Running backprop only on the last time step B) Applying backprop to the unrolled RNN computation graph across all time steps C) Using a different loss function at each time step D) Backpropagating gradients only through the output layer

---

Q13. In BPTT, all hidden states EXCEPT the last one receive gradients from:

A) Only the output $\hat{y}_t$ at that time step B) Only the next hidden state h_{t+1} C) Two sources: from $\hat{y}_t$ AND from h_{t+1} D) Three sources: $\hat{y}_t$, h_{t+1}, and x_t

---

Q14. For very long sequences, full BPTT is not feasible because:

A) The loss function changes B) Unrolling the entire sequence is not feasible and it is not parallelisable C) The embedding matrix becomes too large D) Weight matrices cannot be shared across long sequences

---

Q15. The key advantage of RNN LMs over FFN LMs is:

A) RNN LMs use softmax; FFN LMs do not B) RNN LMs don't have the fixed context window problem; hidden state can represent all preceding words C) RNN LMs are faster to train D) RNN LMs require fewer parameters

---

Q16. The key advantage of RNN LMs over N-gram LMs is:

A) RNN LMs are more interpretable B) RNN LMs don't have the limited context problem of N-gram models C) RNN LMs require less training data D) RNN LMs cannot overfit

---

Q17. RNN Language Models were introduced by:

A) Elman, 1990 B) Bengio et al., 2003 C) Mikolov et al., 2010 D) Rumelhart et al., 1986

Q18. In an RNN LM, words are encoded as:

A) Floating point frequency counts B) One-hot vectors used to look up embeddings in matrix E C) Raw integer indices D) Averaged embeddings of surrounding words

---

Q19. Teacher forcing during RNN LM training means:

A) A teacher supervises the training manually B) Always using the model's own prediction as the next input C) Always using the correct previous word as input rather than the model's prediction D) Only training on sequences where the model is correct

---

Q20. The CE loss for RNN language modelling at time t is:

A) The mean squared error between $\hat{y}_t$ and y_t B) The negative log probability of the correct next word C) The sum of probabilities over the vocabulary D) The dot product of h_t and V

Q21. In RNN LM training, what does the target distribution y_t look like?

A) A uniform distribution over the vocabulary B) A softmax distribution from the previous step C) A one-hot vector where the actual next word has value 1 D) A smoothed probability distribution

---

Q22. Weight tying connects which two matrices?

A) U and W B) W and V C) Embedding matrix E and output matrix V D) U and V

Q23. For weight tying to work, V must be set to:

A) E B) E^T (E transposed) C) E^2 D) $-E$

Q24. Which of the following is NOT a benefit of weight tying?

A) Substantial savings in parameters B) Faster training C) Lower model perplexity D) Elimination of the vanishing gradient problem

---

Q25. In sequence labelling with RNNs:

A) Only the final hidden state is used B) A label is produced at every time step C) No softmax is used D) The embedding matrix is frozen

---

Q26. In autoregressive generation, what is used as the very first input to start generation?

A) A random word embedding B) The end-of-sentence marker </s> C) The beginning-of-sentence marker <s> D) A zero vector

---

Q27. When does autoregressive generation stop?

A) After exactly 100 words B) When the model's loss reaches zero C) When </s> is sampled or a fixed length limit is reached D) When the hidden state reaches zero

---

Q28. What makes neural language models "autoregressive" even though they are not linear?

A) They use linear activation functions B) The word generated at each step is conditioned on the word selected from the previous step C) They use autoregressive loss functions D) They process sequences in reverse

---

Q29. For machine translation using autoregressive generation, what is used to prime the generation?

A) A random start token B) The sentence in the source language C) The beginning-of-sentence marker only D) A pretrained summary of the vocabulary

---

Q30. In sequence classification with RNNs, which hidden state is used to represent the whole sequence?

A) The first hidden state h_1 B) The average of all hidden states C) The final hidden state $h_{\square}$ (or pooled hidden states) D) The hidden state with the highest activation

---

Q31. In sequence classification, where does the loss come from?

A) From every time step B) Only from the final classification output C) From the first and last time steps only D) From the embedding layer

---

Q32. What is end-to-end learning in the context of RNN sequence classification?

A) Training only the output classifier layer B) Training only the embedding layer C) Using the downstream task loss to adjust all weights in the entire model D) Training the RNN and classifier separately

---

Q33. Instead of using only the final hidden state in sequence classification, an alternative is to:

A) Use the first hidden state B) Pool all hidden states using element-wise mean or max C) Average the input embeddings D) Use a separate FFN to re-encode the sequence

---

Q34. Simple RNNs fail on long inputs primarily because of:

A) Insufficient vocabulary size B) Vanishing gradients over many time steps C) Overfitting to short sequences D) Weight matrix size limitations

---

Q35. The modern solution to vanishing gradients in RNNs is:

A) Using larger learning rates B) Using more training data C) Gated architectures such as LSTMs D) Removing the hidden-to-hidden weight matrix W

Q36. In RNN sequence labelling, what is the loss function used?

A) Mean squared error B) Cross-entropy loss C) Hinge loss D) KL divergence

---

Q37. The gradient at the final hidden state h(τ) comes from:

A) Both o(τ) and h(τ+1) B) Only o(τ) — there is no h(τ+1) C) Only W matrix D) The embedding layer

---

Q38. The gradient at h(t) for $t < \tau$ is given by contributions from:

A) $V^T \cdot \nabla \{o(t)\} L$ only B) $W^T \cdot \nabla \{h(t+1)\} L$ only C) $V^T \cdot \nabla \{o(t)\} L + W^T \cdot \nabla \{h(t+1)\} L$ D) $U^T \cdot \nabla \{o(t)\} L + W^T \cdot \nabla \{h(t+1)\} L$

Q39. The gradient at the output logits o(t) is:

A) h□ − y□ B) ŷ(t) − y(t) C) y(t) × ŷ(t) D) softmax(o(t)) only

---

Q40. In an unrolled RNN with 3 time steps, how many copies of the weight matrix W exist in memory?

A) 3 separate copies B) 1 shared matrix used at all 3 steps C) 0 — W is not used in unrolled RNNs D) 2 copies (one for forward, one for backward)

Q41. The probability of the entire word sequence $w_1,...,w$□ in an RNN LM is computed as:

A) The sum of individual word probabilities B) The product of conditional probabilities $P(w_i|w_1,...,w_{i-1})$ C) The softmax of all word probabilities D) The average of all time step losses

---

Q42. The logits vector in an RNN LM (output of V·h□) represents:

A) The final probability distribution B) Scores over the vocabulary before softmax normalisation C) The embedding of the next word D) The hidden state at the next time step

---

Q43. Which application is NOT listed in the slides as using autoregressive RNN generation?

A) Machine translation B) Summarisation C) Question answering D) Part-of-speech tagging

---

Q44. The RNN classifier for sequence classification typically uses what as its classifier?

A) A single logistic regression unit B) A whole 2-layer FFN C) A simple linear layer D) Another RNN

Q45. Which of the following is a listed application of RNNs from the slides?

A) Image segmentation B) Probabilistic language modelling C) Graph classification D) Object detection

---

Q46. TRUE or FALSE: In weight tying, two separate matrices are stored in memory but used at different points.

A) True — two matrices stored, two used B) False — functionally two matrices, but only ONE stored in memory C) True — each matrix is stored separately for gradient computation D) False — no matrices are stored, computed on the fly

---

Q47. In the forward inference equation for RNN LM, the probability of word i being the next word is:

A) The ith element of h□ B) The ith element of the softmax output ŷ□ C) The ith row of embedding matrix E D) The dot product of V and E

Q48. What is the hidden state h(0) at the start of RNN forward propagation?

A) The embedding of the first word B) A specified initial hidden state (typically zeros) C) A random vector resampled each sequence D) The output of the embedding matrix

---

Q49. RNNs are described as having a fairly short short-term memory because:

A) Weight matrices are too small B) Traces of earlier inputs fade when they no longer help predict current output C) The hidden state dimension is fixed D) Gradient clipping removes early information

---

Q50. The self-supervision training approach for RNN LMs means:

A) Human annotators label each word B) The model uses its own predictions as labels C) The correct next word in the corpus serves as the label at every time step D) Labels are generated by another pretrained model

---

Section B — Answers & Explanations

Q1. B — FFNs have fixed-length inputs processed separately. Long-range or discontinuous dependencies outside the window cannot be captured.

Q2. C — Elman (1990) introduced RNNs. Mikolov et al. (2010) introduced RNN language models.

Q3. B — The copy-back mechanism takes the hidden state from t−1 and concatenates it with the current input x□.

Q4. C — U, V, and W are shared (same matrices) across all time steps. This is a defining feature of RNNs.

Q5. C — The inputs x□, hidden states h□, and outputs y□ change at every time step. Weight matrices stay the same.

Q6. C — W connects hidden to hidden: shape [dh × dh].

Q7. A — U connects input to hidden: shape [dh × din].

Q8. C — tanh is the typical activation for hidden state computation in RNNs (slides explicitly state g might be tanh).

Q9. D — Softmax is used at the output layer to produce a probability distribution (f = softmax).

Q10. C — Total loss L = Σ□ L(t), the sum of individual time step losses.

Q11. B — Each step requires the hidden state from the previous step, so steps must be computed sequentially.

Q12. B — BPTT = applying backpropagation to the fully unrolled RNN computation graph across all time steps.

Q13. C — All hidden states except the last receive gradients from both ŷ□ (current output) and h□₊₁ (next hidden state).

Q14. B — For long sequences, unrolling the entire sequence is not feasible, and the process is not parallelisable.

Q15. B — RNN LMs don't have the fixed context window problem. The hidden state can in principle represent all preceding words.

Q16. B — N-gram LMs only have probabilities for seen sequences. RNN LMs can generalise to unseen contexts through the hidden state.

Q17. C — RNN LMs were introduced by Mikolov et al., 2010 (explicitly cited in slides).

Q18. B — Words are encoded as one-hot vectors used to look up their embedding in matrix E.

Q19. C — Teacher forcing: always use the correct previous word (ground truth) as input, not what the model predicted.

Q20. B — CE loss at time t = −log ŷ□[w□] = negative log probability of the correct next word.

Q21. C — Target distribution y□ is a one-hot vector with 1 at the actual next word's position and 0 everywhere else.

Q22. C — Weight tying connects the input embedding matrix E and the output weight matrix V.

Q23. B — $V = E^T$. Since E is [dh × |V|], transposing gives [|V| × dh] which matches the required shape of V.

Q24. D — Weight tying saves parameters, speeds training, and lowers perplexity. It does NOT solve vanishing gradients.

Q25. B — Sequence labelling produces a label/output at every single time step (e.g. POS tagging).

Q26. C — Generation starts with the beginning-of-sentence marker <s>.

Q27. C — Generation stops when </s> is sampled or a fixed length limit is reached.

Q28. B — Though non-linear, the technique is called autoregressive because each generated word is conditioned on the word selected at the previous step.

Q29. B — For machine translation, the source language sentence primes the generation (provides the context).

Q30. C — The final hidden state h□ represents the whole sequence (or pooled hidden states as an alternative).

Q31. B — No intermediate outputs → loss comes only from the final classification. Error is then propagated back through all weights.

Q32. C — End-to-end learning: the downstream task loss propagates back through the classifier AND all the RNN weights.

Q33. B — Alternative to using only the last hidden state: pool all hidden states using element-wise mean or max.

Q34. B — Simple RNNs fail on long inputs primarily due to vanishing gradients — the error signal gets too small to propagate back many steps.

Q35. C — LSTMs (Long Short-Term Memory networks) use gating mechanisms to explicitly decide what to remember and forget, solving the vanishing gradient problem.

Q36. B — CE (cross-entropy) loss is used for sequence labelling, same as for classification tasks.

Q37. B — At the final time step τ, $h(\tau)$ only has $o(\tau)$ as a descendant — no future hidden state exists.

Q38. C — For $t < \tau$: $\nabla_{h(t)} L = V^T \cdot \nabla_{o(t)} L + W^T \cdot \nabla_{h(t+1)} L$ — two gradient sources.

Q39. B — $\nabla_{o(t)}L = \hat{y}(t) - y(t)$, the difference between predicted and true distributions.

Q40. B — W is one shared matrix used at all time steps. Dummy variable notation W(t) is used only for gradient computation bookkeeping.

Q41. B — $P(w_1,...,w_\square) = \Pi\ P(w_i|w_1,...,w_{i-1})$ — chain rule of probability applied to the sequence.

Q42. B — The logits $V \cdot h_\square$ are raw scores over vocabulary before softmax normalisation turns them into probabilities.

Q43. D — POS tagging is sequence labelling, not autoregressive generation. MT, summarisation, and QA are listed as autoregressive generation applications.

Q44. B — The slides state: "the classifier isn't just a single feed-forward layer, but a whole 2-layer FFN."

Q45. B — Probabilistic language modelling is explicitly listed. Image segmentation, graph classification, and object detection are not NLP/RNN tasks covered.

Q46. B — Functionally there are two matrices (E and V), but in memory only ONE matrix is stored. That is the point — memory savings.

Q47. B — $\hat{y}_\square[i]$ = the ith component of the softmax output = probability that word i is the next word.

Q48. B — h(0) is a specified initial hidden state (typically set to zeros) before any input is processed.

Q49. B — Traces of earlier inputs fade over time when that information no longer helps predict the current output — this is the short-term memory limitation.

Q50. C — Self-supervision: the correct next word in the training corpus serves as the gold label at every time step — no manual annotation required.

WEEK 5 — Stacked & Bidirectional RNNs, Attention, LSTMs

---

1. Important Topics

Stacked RNNs

Bidirectional RNNs

LSTMs (gates, context vector, training)

Encoder-Decoder model (inference & training)

Attention mechanism (bottleneck problem, dot-product, bilinear)

---

2. Definitions

Term	Definition
Stacked RNN	Multiple RNNs where the output sequence of one RNN is the input sequence to the next
Bidirectional RNN	Two RNNs run in opposite directions (left-to-right and right-to-left); hidden states concatenated

Forward hidden state (h^f_t)	Hidden state of the left-to-right RNN at time t
Backward hidden state (h^b_t)	Hidden state of the right-to-left RNN at time t
LSTM	Long Short-Term Memory network; extends RNN with a context vector and gates to manage long-range dependencies
Context vector (c)	Additional memory cell in LSTM; stores information across time steps
Forget gate (f)	LSTM gate that decides what information to erase from the previous context state
Add/Input gate (i)	LSTM gate that decides what new information to add to the context state
Output gate (o)	LSTM gate that decides what information from the context state to use for the current hidden state
Gate (mask)	Sigmoid-activated learnable vector applied via element-wise multiplication to control information flow
Encoder	RNN that processes input sequence and produces contextualised hidden representations
Decoder	RNN that takes the context vector and autoregressively generates the output sequence
Context vector c (encoder-decoder)	A fixed representation of the encoder's final hidden state passed to the decoder
Bottleneck	Problem in encoder-decoder where all input information must be compressed into a single context vector
Attention	Mechanism allowing the decoder to access ALL encoder hidden states via a weighted sum, not just the last
Dot-product attention	Simplest attention: score = dot product of decoder and encoder hidden states (similarity-based)
Bilinear attention	Parameterised attention: score = $h^d_{i-1} \cdot W \cdot h^e$; learnable weight matrix W trained end-to-end
Dynamic context vector (c_i)	Attention-based context vector, different at each decoding step; weighted sum of all encoder hidden states

---

3. Key Concepts

Stacked RNNs:

Output sequence of RNN layer 1 → input sequence of RNN layer 2, etc.

Generally outperform single-layer RNNs

Each layer learns representations at different levels of abstraction

More stacks = higher training cost

Optimal number of stacks depends on the application and dataset

Bidirectional RNNs:

Motivation: left-to-right RNNs cannot use future (right) context

Solution: run TWO RNNs — one forward, one backward on reversed input

Combine hidden states: $h_t = [h^f_t ; h^b_t]$ (concatenation) — also element-wise addition or multiplication possible

Forward equations:

$h^f_t = g(Uf \cdot h^f_{t-1} + Wf \cdot x_t)$ ← uses left context

$h^b_t = g(Ub \cdot h^b_{t+1} + Wb \cdot x_t)$ ← uses right context

Introduced by Schuster and Paliwal (1997)

For sequence classification: combine FINAL forward and FINAL backward hidden states

> Solves the problem that the final forward state captures more about the end of the sequence

LSTMs:

Introduced by Hochreiter & Schmidhuber (1997); extended by Gers et al. (2000)

Two key additions over vanilla RNN: (1) context vector c_t, (2) three gates

All three gates share same structure: sigmoid(weighted sum of previous hidden state + current input) → produces a mask [0,1]

Forget gate f_t: mask applied to c_{t-1} → $k_t = f_t \odot c_{t-1}$ (delete irrelevant info)

Add gate i_t: mask applied to g_t (what would be vanilla RNN hidden state) → $j_t = i_t \odot g_t$ (add new info)

Current context: $c_t = k_t + j_t$ (element-wise addition)

Output gate o_t: mask applied to $\tanh(c_t)$ → $h_t = o_t \odot \tanh(c_t)$ (current hidden state)

Both h_t and c_t passed to next time step

Sigmoid used for gates because it pushes outputs toward 0 or 1 (binary-like masking)

LSTMs trained with backprop just like RNNs (unrolled into computation graph)

In practice, LSTMs are the standard unit for modern recurrent systems

Encoder-Decoder:

Three components: Encoder + Context vector c + Decoder

Encoder: processes input, produces hidden states h^e□

Context vector c: typically the final encoder hidden state

Decoder: conditioned on c, autoregressively generates output

Decoder also takes c as additional input at every step (prevents influence waning)

Trained end-to-end with teacher forcing

Training data: pairs of (source, target) sequences separated by a separator token

Loss computed at decoder outputs; gradients propagated back through decoder AND encoder

Pioneered by Kalchbrenner & Blunsom (2013) [CNN encoder + RNN decoder]

Cho et al. (2014) and Sutskever et al. (2014) used RNNs for both encoder and decoder

Attention:

Problem: all encoder information compressed into single context vector c → bottleneck for long sequences; early input information lost

Solution: at each decoder step i, create a dynamic context vector c_i = weighted sum of ALL encoder hidden states

Weights $\alpha_{i,}$□ = how relevant encoder state h^e□ is to current decoder state h^d_{i-1}

Step 1: compute scores (dot-product or bilinear)

Step 2: softmax over scores → normalised weights $\alpha_{i,}$□

Step 3: $c_i = \Sigma$□ $\alpha_{i,}$□ · h^e□ (weighted sum of encoder states)

Dot-product attention: score(i,j) = h^d_{i-1} · h^e□ — requires same dimensionality for encoder and decoder

Bilinear attention: score(i,j) = h^d_{i-1} · W□ · h^e□ — W□ is learned; allows different dimensionalities; more powerful

First developed by Graves (2013) for handwriting recognition; extended by Bahdanau et al. (2015) for MT

Self-attention (used in Transformers) is a more sophisticated form — covered in Week 6

---

4. Key Equations

Bidirectional hidden state:

$h_t = [h^f_t ; h^b_t]$

$h^f_t = g(U^f \cdot h^f_{t-1} + W^f \cdot x_t)$

$h^b_t = g(U^b \cdot h^b_{t+1} + W^b \cdot x_t)$

LSTM gate computations:

$g_t = \tanh(W^g \cdot x_t + U^g \cdot h_{t-1})$ ← what vanilla RNN would compute

$f_t = \sigma(W^f \cdot x_t + U^f \cdot h_{t-1})$ ← forget gate

$i_t = \sigma(W^i \cdot x_t + U^i \cdot h_{t-1})$ ← add/input gate

$o_t = \sigma(W^o \cdot x_t + U^o \cdot h_{t-1})$ ← output gate

$k_t = f_t \odot c_{t-1}$ ← filtered previous context

$j_t = i_t \odot g_t$ ← filtered new info

$c_t = k_t + j_t$ ← new context state

$h_t = o_t \odot \tanh(c_t)$ ← new hidden state

Attention:

Dot-product score: $score(i,j) = h^d_{i-1} \cdot h^e_j$

Bilinear score: $score(i,j) = h^d_{i-1} \cdot W_j \cdot h^e_j$

Normalised weights: $\alpha_i = \text{softmax}(\text{scores})$

Dynamic context: $c_i = \Sigma_j \alpha_{i,j} \cdot h^e_j$

---

5. Things to Remember

Stacked RNNs outperform single-layer but training cost rises with more stacks

Bidirectional RNNs introduced by **Schuster and Paliwal (1997)**

Forward RNN: uses left context only; Backward RNN: uses right context only

Bidirectional for sequence classification: concatenate FINAL forward + FINAL backward hidden states (solves end-bias problem)

LSTM introduced by **Hochreiter & Schmidhuber (1997)**

Three LSTM gates: forget (delete old info), add/input (add new info), output (what to expose as hidden state)

Gates use **sigmoid** (not tanh) → pushes values toward 0 or 1 → binary-like masking

Element-wise multiplication applies the mask

Both h□ AND c□ passed to next LSTM time step

g□ uses **tanh** (same as vanilla RNN hidden state computation)

Encoder-decoder context vector c = final encoder hidden state (the bottleneck)

Attention replaces static c with **dynamic c_i** computed at each decoder step

Dot-product attention: encoder and decoder must have **same dimensionality**

Bilinear attention: allows **different dimensionalities**; W□ is **learnable**

Attention first: Graves (2013); extended for MT: **Bahdanau et al. (2015)**

Encoder-decoder first: **Kalchbrenner & Blunsom (2013)**; both RNNs: **Cho et al. (2014) / Sutskever et al. (2014)**

Transformers use **self-attention** — covered in Week 6

WEEK 5 — MCQs

Section A — Questions

Q1. In a stacked RNN, what serves as the input to each successive RNN layer?

A) The weight matrices from the previous layer B) The entire sequence of outputs from the previous RNN layer C) Only the final hidden state of the previous layer D) The original input repeated for each layer

---

Q2. Why do stacked RNNs generally outperform single-layer RNNs?

A) They use more training data automatically B) They apply dropout at every layer C) Different layers learn representations at different levels of abstraction D) They eliminate the vanishing gradient problem

---

Q3. What is the main limitation of a standard left-to-right RNN?

A) It cannot process sequences longer than 100 tokens B) It cannot use any future (right) context C) It requires bidirectional weight matrices D) It uses the wrong loss function

---

Q4. Bidirectional RNNs were introduced by:

A) Hochreiter and Schmidhuber, 1997 B) Elman, 1990 C) Schuster and Paliwal, 1997 D) Bahdanau et al., 2015

Q5. In a bidirectional RNN, how are the forward and backward hidden states most commonly combined?

A) Element-wise multiplication only B) Subtraction C) Concatenation [h^f_t ; h^b_t] D) Averaging only

---

Q6. The backward hidden state h^b_t is computed using:

A) h^b_{t-1} (previous backward state) and x_t B) h^b_{t+1} (next backward state) and x_t C) h^f_t (forward state) and x_t D) Only the current input x_t

---

Q7. For bidirectional RNN sequence classification, which hidden states are used as input to the classifier?

A) All hidden states averaged together B) Only the final forward hidden state C) The final forward AND final backward hidden states (combined) D) The first forward and first backward hidden states

---

Q8. Why is combining the final forward and backward hidden states beneficial for sequence classification?

A) It doubles the learning rate B) Forward RNN captures more about the sequence end; backward RNN captures more about the start C) It reduces the number of parameters D) It eliminates the need for a softmax layer

---

Q9. LSTMs were introduced by:

A) Elman, 1990 B) Mikolov et al., 2010 C) Hochreiter and Schmidhuber, 1997 D) Bahdanau et al., 2015

Q10. What are the two key additions LSTMs make over vanilla RNNs?

A) More layers and larger weight matrices B) A context vector c□ and three gates C) Bidirectional processing and dropout D) Softmax output and larger embeddings

---

Q11. Why is sigmoid used as the activation function for LSTM gates rather than tanh?

A) Sigmoid is faster to compute B) Sigmoid pushes outputs toward 0 or 1, making it behave like a binary mask C) Sigmoid avoids vanishing gradients D) Sigmoid produces negative values needed for forgetting

---

Q12. The forget gate in an LSTM is applied to:

A) The current input x□ B) The current candidate state g□ C) The previous context state c□$_{-1}$ D) The current hidden state h□

Q13. The add/input gate in an LSTM is applied to:

A) The previous context state c□$_{-1}$ B) The output gate C) g□ — what would be the hidden state in a vanilla RNN D) The current context state c□

Q14. The current LSTM context state c□ is formed by:

A) Multiplying k□ and j□ B) Adding k□ and j□ element-wise C) Concatenating k□ and j□ D) Passing k□ through sigmoid

---

Q15. The LSTM hidden state h□ is computed as:

A) tanh(c□) alone B) o□ ⊙ c□ C) o□ ⊙ tanh(c□) D) f□ ⊙ tanh(c□)

Q16. What two values does an LSTM pass to the next time step?

A) h□ and g□ B) c□ and g□ C) h□ and c□ D) f□ and o□

Q17. In an LSTM, g□ (the candidate hidden state) uses which activation function?

A) Sigmoid B) ReLU C) Softmax D) tanh

Q18. Intuitively, which LSTM gate decides what information from past context is no longer needed?

A) Add/input gate B) Output gate C) Forget gate D) Context gate

---

Q19. Which LSTM gate controls what information from the context state is used for the current hidden state vs. preserved for future steps?

A) Forget gate B) Output gate C) Add/input gate D) Context gate

---

Q20. How are LSTMs trained?

A) With a completely different algorithm from RNNs B) Unrolled into feedforward networks and trained with backpropagation C) Using forward-only passes with no gradient computation D) Using reinforcement learning

---

Q21. The three components of an encoder-decoder network are:

A) Embedding layer, hidden layer, output layer B) Encoder, context vector c, decoder C) Forward RNN, backward RNN, classifier D) Input gates, hidden gates, output gates

---

Q22. In the simplest RNN encoder-decoder, what serves as the context vector c?

A) The average of all encoder hidden states B) The first encoder hidden state C) The final encoder hidden state D) A learned static vector

---

Q23. The first encoder-decoder model was pioneered by:

A) Bahdanau et al., 2015 B) Kalchbrenner and Blunsom, 2013 C) Mikolov et al., 2010 D) Hochreiter and Schmidhuber, 1997

Q24. Cho et al. (2014) and Sutskever et al. (2014) showed how to use:

A) CNNs for both encoder and decoder B) Transformers for both encoder and decoder C) Extended RNNs for both encoder and decoder D) Bidirectional RNNs as decoders only

---

Q25. Why does the decoder in an encoder-decoder receive c as additional input at every time step?

A) To increase parameter count B) To prevent the context's influence from waning over time C) To replace the need for teacher forcing D) To compute attention weights

---

Q26. Encoder-decoder training uses which technique for the decoder's forward pass?

A) Autoregressive sampling from the model's own predictions B) Teacher forcing — correct previous output used as next input C) Random token insertion D) Beam search

---

Q27. During encoder-decoder training, where does gradient propagation go?

A) Only through the decoder B) Through the decoder and back through the encoder C) Only through the encoder D) Only through the context vector c

---

Q28. What is the bottleneck problem in vanilla encoder-decoders?

A) The decoder is too slow B) All input information must be compressed into a single fixed-length context vector c C) The encoder cannot handle long sequences D) The attention weights are not normalised

---

Q29. How does attention solve the bottleneck problem?

A) By increasing the size of the context vector B) By allowing the decoder to access all encoder hidden states via a weighted sum at each step C) By adding more encoder layers D) By using bidirectional encoding

---

Q30. In attention, the weights $\alpha_{i,\square}$ represent:

A) The learning rate for step j B) The proportional relevance of encoder hidden state $h^{e}\square$ to decoder state h^{d}_{i-1} C) The gradient of the loss at step j D) The output probability of word j

---

Q31. What operation is applied to raw attention scores to produce the final weight distribution?

A) tanh B) Sigmoid C) Softmax D) ReLU

Q32. The dynamic context vector c_i in attention is computed as:

A) The encoder's final hidden state B) $\Sigma\square\ \alpha_{i,\square} \cdot h^{e}\square$ — weighted sum of all encoder hidden states C) The dot product of h^{d}_i and h^{e}_i D) The average of all encoder hidden states

---

Q33. Dot-product attention computes the score as:

A) $h^d_{i-1} \cdot W\square \cdot h^e\square$ B) sigmoid($h^d_{i-1} \cdot h^e\square$) C) $h^d_{i-1} \cdot h^e\square$ (dot product) D) $||h^d_{i-1} - h^e\square||$

Q34. What is a key limitation of dot-product attention compared to bilinear attention?

A) Dot-product attention is not differentiable B) Dot-product attention requires the encoder and decoder to have the same dimensionality C) Dot-product attention cannot be normalised with softmax D) Dot-product attention requires a separate training phase

---

Q35. Bilinear attention computes the score as:

A) $h^d_{i-1} \cdot h^e\square$ B) $h^d_{i-1} \cdot W\square \cdot h^e\square$ where $W\square$ is a learnable matrix C) sigmoid($h^d_{i-1} + h^e\square$) D) softmax($h^d_{i-1} \times h^e\square$)

Q36. The attention weight matrix $W\square$ in bilinear attention is:

A) Fixed and not trained B) Initialised to the identity matrix and kept fixed C) Trained during normal end-to-end training D) Shared with the encoder weight matrix

---

Q37. Which of the following is an advantage of bilinear attention over dot-product attention?

A) Bilinear attention is computationally simpler B) Bilinear attention allows encoder and decoder hidden states of different dimensionality C) Bilinear attention doesn't require softmax normalisation D) Bilinear attention uses fewer parameters

---

Q38. Attention was first developed by:

A) Bahdanau et al., 2015 for MT B) Graves (2013) for handwriting recognition C) Cho et al., 2014 for MT D) Hochreiter & Schmidhuber, 1997 for LSTMs

---

Q39. Attention was extended for machine translation by:

A) Graves, 2013 B) Sutskever et al., 2014 C) Bahdanau et al., 2015 D) Kalchbrenner and Blunsom, 2013

Q40. What type of attention is used in Transformers (to be covered in Week 6)?

A) Dot-product attention B) Bilinear attention C) Self-attention D) Forget-gate attention

---

Q41. In an LSTM, gates are implemented as learnable masks applied via:

A) Matrix multiplication B) Element-wise multiplication (⊙) C) Element-wise addition D) Dot product

---

Q42. TRUE or FALSE: In a bidirectional RNN, the forward and backward RNNs share the same weight matrices (Uf = Ub, Wf = Wb).

A) True — weight sharing reduces parameters B) False — they are two independent RNNs with separate weight matrices C) True — they are trained jointly on the same data D) False — only the hidden-to-hidden weights are shared

---

Q43. In the encoder-decoder architecture, which types of networks can serve as the encoder?

A) Only RNNs B) LSTMs, convolutional networks, and Transformers C) Only FFNs D) Only bidirectional RNNs

---

Q44. In vanilla encoder-decoder RNNs, information at the beginning of long sequences may be poorly represented because:

A) The encoder stops after a fixed number of steps B) The final hidden state must compress all information and early information may not survive C) The decoder uses teacher forcing which ignores early context D) The embedding matrix doesn't cover early words

---

Q45. The dynamic context vector in attention differs from the static context vector in vanilla encoder-decoders in that:

A) It is computed only once during training B) It is the same at every decoding step C) It is computed freshly at each decoding step based on all encoder hidden states D) It does not use encoder hidden states

---

Q46. In the LSTM, k_t is defined as:

A) $i_t \odot g_t$ B) $f_t \odot c_{t-1}$ C) $o_t \odot \tanh(c_t)$ D) $f_t \odot h_{t-1}$

Q47. In the LSTM, j_t is defined as:

A) $f_t \odot c_{t-1}$ B) $o_t \odot \tanh(c_t)$ C) $i_t \odot g_t$ D) $f_t \odot g_t$

Q48. Why do LSTMs address the vanishing gradient problem better than vanilla RNNs?

A) LSTMs use ReLU instead of tanh throughout B) The context vector and gates allow gradients to flow through c_t without being repeatedly multiplied by W at every step C) LSTMs use larger hidden state dimensions D) LSTMs apply dropout automatically

---

Q49. For sequence labelling using bidirectional RNNs, the output at each time step is based on:

A) Only the forward hidden state B) Only the backward hidden state C) The concatenated forward and backward hidden states at that time step D) The final forward hidden state only

---

Q50. LSTMs in practice have become:

A) Rarely used due to computational cost B) The standard unit for any modern system making use of recurrent networks C) Only useful for speech recognition tasks D) Replaced entirely by vanilla RNNs for most tasks

---

Section B — Answers & Explanations

Q1. B — In stacked RNNs, the entire output sequence of one RNN layer feeds as input to the next layer.

Q2. C — Each layer in a stacked RNN learns representations at a different level of abstraction, increasing representational power.

Q3. B — Standard left-to-right RNNs only use left (past) context. Future (right) context is unavailable.

Q4. C — Bidirectional RNNs were introduced by Schuster and Paliwal (1997), as stated in the slides.

Q5. C — The most common combination is concatenation: $h_t = [h^f_t ; h^b_t]$. Addition and multiplication are alternatives but concatenation is standard.

Q6. B — The backward RNN processes right-to-left, so h^b_t depends on h^b_{t+1} (the next backward state) and x_t.

Q7. C — For sequence classification, the final forward AND final backward hidden states are combined (e.g. concatenated) as input to the classifier.

Q8. B — The forward RNN naturally captures more end-of-sequence information; the backward RNN captures more beginning-of-sequence information. Together they provide a balanced representation.

Q9. C — LSTMs introduced by Hochreiter and Schmidhuber (1997), extended by Gers et al. (2000).

Q10. B — LSTMs add a context vector c_t and three gates (forget, add/input, output) over vanilla RNNs.

Q11. B — Sigmoid pushes outputs toward 0 or 1, making it behave like a soft binary mask — letting information through or blocking it.

Q12. C — The forget gate f_t is applied to the PREVIOUS context state c_{t-1}: $k_t = f_t \odot c_{t-1}$.

Q13. C — The add/input gate i_t is applied to g_t (the candidate hidden state, i.e. what vanilla RNN would compute): $j_t = i_t \odot g_t$.

Q14. B — $c_t = k_t + j_t$ — element-wise addition of filtered old context and filtered new information.

Q15. C — $h_t = o_t \odot \tanh(c_t)$. The output gate masks the tanh-transformed context state.

Q16. C — Both h_t (hidden state) and c_t (context state) are passed to the next time step in an LSTM.

Q17. D — $g_t = \tanh(W_g \cdot x_t + U_g \cdot h_{t-1})$. tanh is used for the candidate hidden state, same as vanilla RNN.

Q18. C — The forget gate explicitly controls what past context to erase — intuitively, it "forgets" irrelevant information.

Q19. B — The output gate controls what from the current context c_t is exposed as the hidden state h_t versus what stays in c_t for future steps.

Q20. B — LSTMs are unrolled into feedforward networks and trained with backpropagation, exactly like vanilla RNNs.

Q21. B — The three components are: encoder, context vector c, and decoder.

Q22. C — In the simplest form, c = final encoder hidden state (the bottleneck).

Q23. B — Kalchbrenner and Blunsom (2013) pioneered encoder-decoder models (CNN encoder + RNN decoder).

Q24. C — Cho et al. (2014) and Sutskever et al. (2014) showed how to use extended RNNs for BOTH encoder and decoder.

Q25. B — c is provided at every decoder step to prevent its influence from waning — otherwise the decoder might "forget" the input as generation progresses.

Q26. B — Teacher forcing: during training, the correct (gold) output from the previous step is used as the current decoder input.

Q27. B — Gradients propagate backward through the decoder and all the way back through the encoder in end-to-end training.

Q28. B — The bottleneck: ALL input information must be compressed into the single fixed-length final encoder hidden state c. Early information in long sequences is poorly preserved.

Q29. B — Attention allows the decoder to take a weighted sum of ALL encoder hidden states at each step, not just the last one.

Q30. B — $\alpha_{i,\square}$ = proportional relevance of encoder state $h^{e}\square$ to decoder state h^{d}_{i-1} at decoding step i.

Q31. C — Softmax normalises the raw scores into a probability distribution (summing to 1) to give the final attention weights α_i.

Q32. B — $c_i = \Sigma\square\ \alpha_{i,\square} \cdot h^{e}\square$ — a weighted sum of all encoder hidden states, weights given by α_i.

Q33. C — Dot-product attention: score(i,j) = $h^{d}_{i-1} \cdot h^{e}\square$. Similarity between decoder and encoder hidden states.

Q34. B — Dot-product attention requires encoder and decoder hidden states to have the same dimensionality (for the dot product to be defined).

Q35. B — Bilinear attention: score(i,j) = $h^{d}_{i-1} \cdot W\square \cdot h^{e}\square$ where $W\square$ is a learnable weight matrix.

Q36. C — $W\square$ is trained during normal end-to-end training, allowing the model to learn what relevance function works best.

Q37. B — Bilinear attention allows encoder and decoder to have different dimensionalities (because W□ can map between spaces of different sizes).

Q38. B — Attention was first developed by Graves (2013) for handwriting recognition.

Q39. C — Bahdanau et al. (2015) extended attention for machine translation.

Q40. C — Transformers use self-attention — a more sophisticated form of attention, to be covered in Week 6.

Q41. B — Gates are applied via element-wise multiplication (⊙), not matrix multiplication. This allows selective filtering.

Q42. B — False. Forward and backward RNNs are two independent networks with their own separate weight matrices (Uf, Wf for forward; Ub, Wb for backward).

Q43. B — The slides explicitly state: "LSTMs, convolutional networks, and Transformers can all be employed as encoders."

Q44. B — The final hidden state must compress all sequence information. For long sequences, early information may be lost or poorly represented by the time the final state is reached.

Q45. C — The dynamic context c_i is computed freshly at each decoding step i, based on all encoder hidden states weighted by their relevance to the current decoder state.

Q46. B — k□ = f□ ⊙ c□$_{-1}$: the forget gate applied to the previous context state.

Q47. C — j□ = i□ ⊙ g□: the add/input gate applied to the candidate hidden state g□.

Q48. B — The context vector c□ provides a path for gradients to flow through time with fewer repeated multiplications by the hidden-to-hidden weight matrix W, reducing vanishing gradient severity.

Q49. C — In bidirectional RNN sequence labelling, outputs at each step use the concatenated forward and backward hidden states at that specific time step.

Q50. B — The slides explicitly state: "LSTMs rather than RNNs have become the standard unit for any modern system that makes use of recurrent networks."