1.

a)

i)

Momentum effect is trying to avoid oscillation, by leveraging the fact that it preserves both the previous speed that our weights have been updating and the new gradient. By using the previous speed as exponential averaging it reduce varying so much. In other words, "m" has less effect in updating gradient. Therefore, θ oscillates less and reaching to global minima become faster and more efficient.

***ii*)**

in Adam case, small gradient parameter will receive larger updates. Let's explain that in this way :

by dividing the update parameter to √v, whenever we have large √v the learning rate will reduce a lot which shows that when we have large updates our model shouldn't be affected in that way and should preserve its stability. And whenever √v is small and our model isn't updating its weights we should in some way push it so dividing by √v which is small value, makes updating weight faster.

b)

i)

as said in question γ is used to have the expected value of h_{drop} is still h

$$E_{P\text{-}drop}[h_{drop}]_i = h_i$$

So we have some units which are canceled and become zero with probability p_{drop}

And other term which remains and those are presereved with probability $1\text{-}p_{drop}$ *. the calculation will be:*

$$E_{P\text{-}drop}[h_{drop}]_i = p_{drop} * h + (1\text{-}p_{drop}) * h_i = (1\text{-}p_{drop}) * h_i \ .$$

This term is zero

So we should somehow cancel $1\text{-}p_{drop}$ *. here* γ *will help us if we have:*

$$\gamma = 1 / (1 - p_{drop})$$

so meets are needed. We have expected value of h_{drop} *equal to h. this calculation is called inverse dropout. Which during test time we want neurons unchanged therefore in training time this term is multiplied to neutralize effect of removing neuron*

ii)

dropout is used for reducing the effect of overfitting(regularization). So in training time we want some neurons to be *Off* so that our model does not rely on every neuron. And effect of some neurons are applied therefore by leveraging on specific percent of our neurons all of them are not used and our model doesn't get overfitted on our training set. But in test time we want all of our model information which has learned through training phase be used so that our model performance gets better and that's why we need all of our neuron and none of them should be dropped.

2.

a)

Stack	Buffer	New Dependency	Transition
[ROOT]	[I, attended, lectures, in, the, NLP, class]		Initial Configuration
[ROOT, I]	[attended, lectures, in, the, NLP, class]		SHIFT
[ROOT, I, attended]	[lectures, in, the, NLP, class]		SHIFT
[ROOT, attended]	[lectures, in, the, NLP, class]	attended→I	LEFT-ARC
[ROOT, attended, lectures]	[in, the, NLP, class]		SHIFT
[ROOT, attended]	[in, the, NLP, class]	attended→lectures	RIGHT-ARC
[ROOT, attended, in]	[the, NLP, class]		SHIFT
[ROOT, attended, in, the]	[NLP, class]		SHIFT
[ROOT, attended, in, the, NLP]	[class]		SHIFT
[ROOT, attended, in, the, NLP, class]	[]		SHIFT
[ROOT, attended, in, the, class]		class→NLP	LEFT-ARC
[ROOT, attended, in, class]		class→the	LEFT-ARC
[ROOT, attended, class]		class →in	LEFT-ARC
[ROOT, attended]		attended →class	RIGHT-ARC
[ROOT]		ROOT →attended	RIGHT-ARC

b)

In fact, in each iteration either we are pushing some words in our stack (SHIFT) OR we are popping them out with some action (LEFT-ARC, RIGHT-ARC). For each word we are adding to our stack it needs to be deleted in some other iteration. So if we have ***N*** words in our sentence it needs ***N*** time to be pushed to our stack and ***N*** time to be popped. So in all we need ***2N*** steps.

e)

```
New best dev UAS! Saving model.

Epoch 9 out of 10
100%|
Average Train Loss: 0.0684912702959692
Evaluating on dev set
1445850it [00:00, 58991006.30it/s]
- dev UAS: 88.57
New best dev UAS! Saving model.

Epoch 10 out of 10
100%|
Average Train Loss: 0.06556430490469778
Evaluating on dev set
1445850it [00:00, 53544897.34it/s]
- dev UAS: 88.73
New best dev UAS! Saving model.

================================================================================
TESTING
================================================================================
Restoring the best model weights found on the dev set
Final evaluation on test set
2919736it [00:00, 78898183.07it/s]
- test UAS: 89.20
Done!
```

f)

i)

- **Error type**: Verb Phrase Attachment Error
- **Incorrect dependency**: acquisition → citing
- **Correct dependency**: blocked → citing

ii) this sentence has two parsing mistakes

- **Error type**: Modifier Attachment Error
- **Incorrect dependency**: had → already
- **Correct dependency**: left → already

- **Error type**: Modifier Attachment Error
- **Incorrect dependency**: left → early
- **Correct dependency**: afternoon → early

iii)

- **Error type**: Prepositional Phrase Attachment Error
- **Incorrect dependency**: declined → decision
- **Correct dependency**: reasons → decision

iv)

- **Error type**: Coordination Attachment Error
- **Incorrect dependency**: affects → one
- **Correct dependency**: plants → one

g)

Sometimes, a word can have multiple meanings and there is **ambiguity** which meaning of that word is used in the context. So part-of-speech(POS) tagging resolves that. It gives us the POS of a word for example it's noun or verb (for example book) and gives us the grammatical structure of sentence, so through this understanding we can now have the meaning of that word and make our dependency parser more easily (which word depends on the other word). By using that we are able to distinguish which next action should our neural network model take, if our model knows that POS of that word.