T alex1a.

Model A is overfitting

Model B is underfitting

Model C is performing regression correctly and would generalize well on test data.

1bi.

As the loss in A is decreasing much more quickly and is more stable, A is likely training on more data per iteration and using full batch descent.

Whereas setting B is trained on minibatches and so is much noisier per iteration. Though setting B would likely converge quicker if the x-axis was time.

(Answer provided by comments)

1bii.

(loss is log scale) (Answer from comments)

1biii.

A, faster convergence to the minimum and frequent oscillations around this minimum once reached. The high LR is causing it to jiggle around the optimum.

Alternative answer (From comments): A has a higher LR due to the faster decrease of loss. As we use the entire dataset each iteration, we can be more confident the gradient direction is correct and therefore use a bigger LR

(Volatility in B is just due to SGD)

1biv.

The model reaches a local/global minimum, The learning rate is larger than the distance between the loss and the optimal loss and so it circles around the optimum.

1ci.

Yes. ReLU sets the value of any nodes less than 0 to 0. This “turns off” the nodes and they are no longer used in computation. Therefore during backpropagation the network only trains nodes relevant to the computation, thereby making it more efficient.

Additionally, ReLU is much less vulnerable to the vanishing gradient problem than tanh. Therefore, gradient information is propagated better to earlier layers and the network trains faster.

1cii.

Each of the neural network's weights receives an update proportional to the [partial derivative](https://en.wikipedia.org/wiki/Partial_derivative) of the error function with respect to the current weight in each iteration of training. The deeper the network is, the more partial gradients are multiplied together, reducing them and resulting in the effect known as “vanishing gradients”

The Tanh activation function has small gradients (saturation) when it outputs values close to -1 and 1, thereby accelerating the vanishing gradient effect. ReLU allows gradients to pass through unmodified (if the output was positive), thereby not changing the gradient. This avoids saturation of the gradient.

1ciii.

Xavier weight initialization (Initialize weights such that initial gradients unlikely to be small)

Leaky ReLU (Allow small gradients for negative values)

Residual connections (propagate the gradient backwards through skip connections)

1di.

(Here the tanh/softmax layer refers to linear neurons + the activation)

Answer (from comments):wk shows a bottleneck at 16 neurons, which means it reduces the data to a space of dimension 16, so it is less general than the first one.

The linear multiplication matrix of model B is a map of rank 16 at most, so it has less representation power than the rank 64 map of model A.

1dii.

Model A 128\*64=8192

Model B (128 \* 16) + (16 \* 64) =3072

1diii.

Model A

1div.

Model B will train faster because there are less parameters?

The bottleneck of 16 in Model B may force the network to generalise better to unseen data, since it cannot fit too closely to specific characteristics of the training examples.

2ai.

|  |  |  |
| --- | --- | --- |
| 4 | 5 | 4 |
| 7 | 7 | 6 |
| 7 | 7 | 6 |

2aii.

You have to flip the kernel horizontally and vertically first to do a convolution.

|  |  |  |
| --- | --- | --- |
| 2 | 2 | 0 |
| 5 | 6 | -3 |
| 8 | 4 | 3 |

For exam, in case someone hasn’t yet prepared a colab notebook with all functions 😛  
input = torch.FloatTensor([[1, 0, 1, 2, 1, 0, 0], ...])

input = input.view(1, 1, input.size(0), input.size(1))

weight = torch.FloatTensor([[0, 1, 0], ...])

weight = weight.flip([0, 1])

weight = weight.view(1, 1, weight.size(0), weight.size(1))

F.conv2d(input, weight, bias=None, stride=2, padding=0)

2bi. (“zero padding” = padding with zeros) (NOT “no” padding)

(M – m + 1) by (N – n + 1)

2bii.

(M-m)/2 + 2 by (N-n)/2 + 2 (from comments) (assuming padding of size 1)

-> The formulation is more likely to imply that the padding is (m-1)/2 by (n-1)/2, so that the passing counteracts the effect of the filter crop. That would make the outputs (M+1)/2 by (N+1)/2

Correction (I think, because there is 0 padding not 1)

(M-m)/2 + 1 by (N-n)/2 + 1

2biii.

m \* n \* (M – m + 3) \* ( N - n + 3) (assuming padding of size 1) (from comments)

If the zero-padding is used to preserve image size: M \* N \* m \* n

Correction (I think, because there is 0 padding not 1)

(M – m + 1)\* (N – n + 1) \*m\*n

2biv.

m \* (M – m + 3) \* N + n \* (N – n + 3) \* (M - m + 3)

If the zero-padding is used to preserve image size: M \* N \* m + M \* N \* n

Correction (I think, because there is 0 padding not 1)

m \* (M – m + 1) \* N + n \* (N – n + 1) \* (M - m + 1)

2bv.

Yes. You are simply breaking the m x n equation into 2 smaller equations and then added them. Either way it is the same formula, just rearranged.

Convolutions are commutative and associative.

2bvi.

No. You have added non-linearity to only part of the equation. This breaks the link between the 2.

2bvii.

No. The rank of the convolution matrix must equal 1. (Columns must be linearly dependent)

2bviii.

|  |
| --- |
| 1 |
| 1 |
| 1 |
| 1 | | 1 | 1 |

|  |
| --- |
| 1 |
| 0 |
| -1 |
| 1 | | 2 | 1 |

Not possible, columns 1 and 2 are linearly independent.

2ci.

Pooling layers have several roles:

(a) Approximate shift invariance: They allow flexibility in the location of certain features in the image, therefore allowing non-rigid or rotated or scaled objects to be recognized,

(b) they allow unions of features to be computed, e.g. blue eyes or green eyes. (More correctly, pooling forces the network to learn high level features)

(c) they reduce the output image size.

2cii.

Fewer calculations whilst still reducing the image size.

More complexity in strided convolution vs max pooling (which could be considered a fixed convolution)

Max pooling breaks shift equivariance

No loss of data

3ai. (DON’T THINK IT IS ASSESSED THIS YEAR)

I want to say feedback loop or hidden state?

Sets the node as a closed loop. This allows the node to recompute its value with the additional input of its output.

3aii.

I want to say feedback loop or hidden state?

This allows the node to recompute the earlier nodes value with the additional input of its output. The second hidden layer must wait for the node in the first layer to be recomputed before it can finish.

3aiii.

I want to say feedforward weight?

The output of the 3rd node in the 2nd layer is passed as an input to the 2nd node in the 2nd layer. The layer cannot be computed in parallel.

3bi.

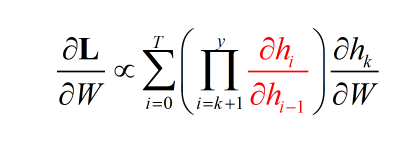
Backpropagation Through Time, or BPTT, is the application of the Backpropagation training algorithm to recurrent neural network applied to sequence data like a time series. A recurrent neural network is shown one input each timestep and predicts one output.

Between time steps we pass a hidden state that contains the important information from the previous time steps.

3bii.

We compute the Loss at each timestep. The loss gradient is passed backwards between timesteps via the chain rule and the loss per timestep is the weighted loss of its future time steps. We sum up the contributions of each time step to the gradient and update the weights. Note that this is the same as the standard backpropagation algorithm that we use in deep FF Networks. The key difference is that we sum up the gradients for W at each time step.

3biii.





In the first case the term goes to 0 exponentially fast, in the second the term goes to infinity exponentially fast.

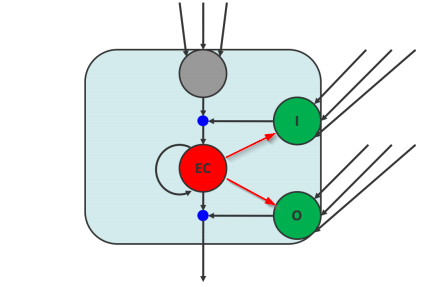
3biv.

Gradient Clipping

Gating (using GRU or LSTM)

Truncating the BPTT

3ci.



3cii.

3ciii.

Output gate,

GRU is easier to modify than an LSTM, it also trains quicker. LSTM is more accurate on dataset using longer sequences (not really though)

3d.

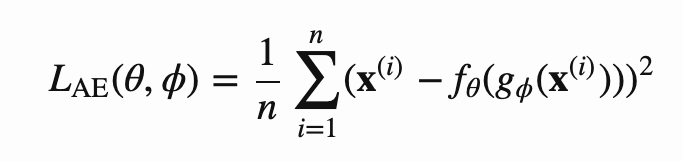
Not examinablet

4ai.

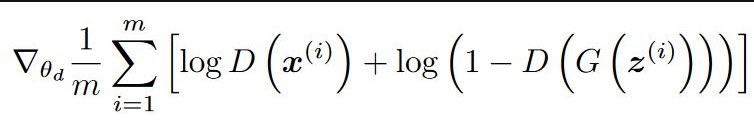
VAEs deconstruct their input into latent parameters (mean, variance) and then attempt to reconstruct their original input. This can be modified to reconstruct alternatives like segmented representations.

GANs attempt to build realistic images from latent noise variables (mean, variance) using a generator network. An adjoining Discriminator network tries to discriminate real images from those built by the Generator. The Generator and Discriminator are trained to optimise a two-player minimax objective.

VAE- this is AE not VAE ->



GAN Discriminator



GAN Generator

log(D(G(z)))

4aii.

Conditional Gan

The network takes the label as an input as well

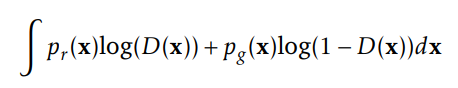
Loss function is the same but with x and z conditoned on y

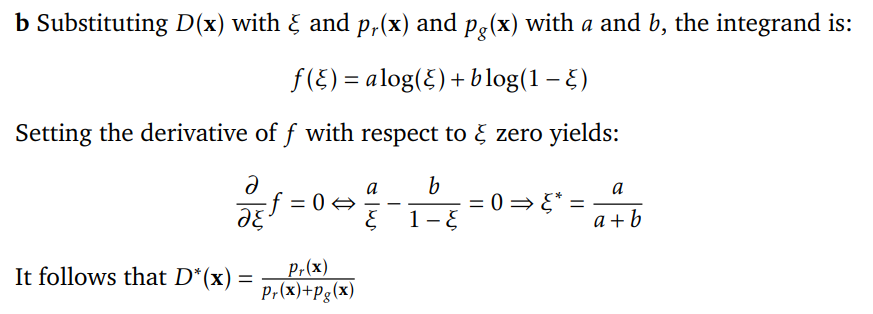
4b.

ne The optimal discriminator will select every real image as real and every fake image as fake.

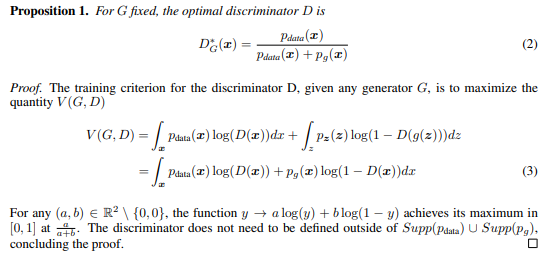
See tutorial 6 question 1.

We have the loss function of D(x) being





Proof from the original paper:



pd = pg for optimal generator, so probability = 0.5