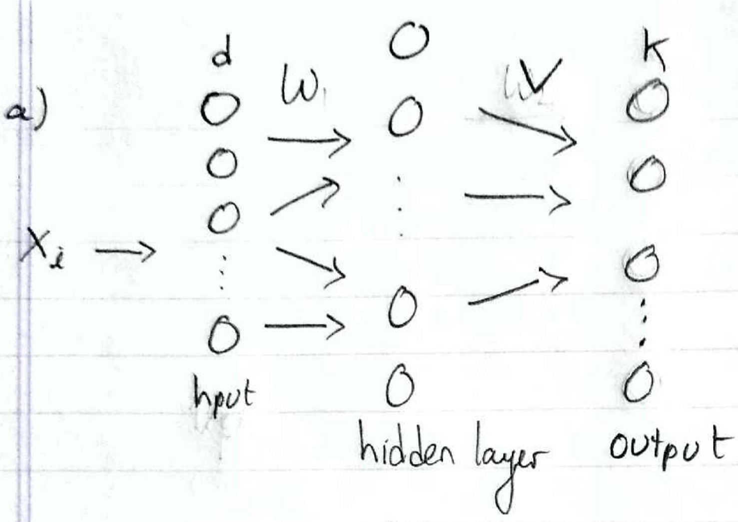
3a)



Input $\quad x \in \mathbb{R}^d$
Output $\quad y \in \mathbb{R}^k$

MLP: $\quad a(x) = W_1^T x + b_1$, hidden layer pre-activation, b_1 is the bias

$h(x) = g(a(x))$, hidden layer activation w/ activation function g.

$\hat{y} = f(x) = o(V^T h(x) + b_2)$, bias b_2

output activation function

Parameters $W \in \mathbb{R}^{d \times z}$

$V \in \mathbb{R}^{z \times k}$

$b_1 \in \mathbb{R}^z$

$b_2 \in \mathbb{R}^k$

Parameters: $\quad d \cdot z + z \cdot k + z + k$

where z is the number of nodes in the hidden layer.

Considerations:

- Since $y_i \in \mathbb{R}^k$, a 1-of-k encoding for the class of training sample i, the output activation function should be the softmax function

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{k} e^{z_k}}$$

This will yield a probability distribution for the output

- Verify our usage of only one hidden layer. If the data is linearly separable, we don't need any hidden layers. This reduces the complexity of our model and reduces tendency of our model to overfit (as we have less parameters)

1 hidden: $d \cdot z + z \cdot k + z + k \longrightarrow \underset{\text{weights + bias}}{d \cdot k + k}$

1 hidden layer can approximate any function that contains a continuous mapping from one finite space to another. Is this enough?
If we extend to 2 hidden layers, this increases the complexity ($\Longrightarrow$ more variance) of our model.

- $g(\cdot)$, our hidden layer activation function.
↳ can be, for ex. $\sigma(a) = \frac{1}{1+e^{-a}}$, $\tanh(a) = \frac{e^{2a}-1}{e^{2a}+1}$

- The number of hidden units of our hidden layer is another consideration.

Increasing the hidden layer by 1 node

$$\Rightarrow \; d*(z+1) + (z+1)h + (z+1) + h$$
$$\underline{-(dz + zh + z + h)}$$
$$d + k + 1$$

Increase the number of parameters we have by $(d+h+1)$. Thus, if we have many features and many classes, an increase of 1 hidden unit significantly increases the number of params and thus, the complexity of our model.
$\Rightarrow$ the variation would increase.

- We have other hyperparameter considerations, such as learning rate, momentum, etc.

- We have exploding and vanishing gradient problems to consider (possible sol: ReLu, batch normalization, regularization)

b) i) $E = -\sum_{i=1}^{I} \sum_{j=1}^{J} a_{i,j} \log(b_{i,j})$

I := sample number (i.e

J := class number (i.e number of classes)

$a_{i,j}$:= For sample i, $a_i \in \mathbb{R}^J$, a one-hot encoding
$a_{i,j} = 1$ if class j is the correct label for sample i (=0 otherwise)

$b_{i,j}$:= predicted probability sample i is from class j.

ii) Consider $J=2$ (2 classes), let $\{x_i, y_i\}_{i=1}^{n}$ be dataset

Binary Cross Entropy $= -\sum_{i=1}^{n} y_i \log p_\theta(y|x_i) + (1-y_i)\log[1-p_\theta(y|x_i)]$

Consider Bernoulli

$$p(y|\pi) = \prod_{i=1}^{n} \pi_i^{y}(1-\pi_i)^{1-y_i}$$

$$\Rightarrow p(y|x,\theta) = \prod_{i=1}^{n} p_\theta(y|x_i)^{y_i}(1-p_\theta(y|x_i))^{1-y_i}, \quad p_\theta(y|x_i) = \sigma(\theta^T x) = \frac{1}{1+e^{-\theta x}}$$

Taking the log of this

$$\log(p(y|x,\theta)) = \sum_{i=1}^{n} y_i \log(p_\theta(y|x_i)) + (1-y_i)\log(1-p_\theta(y|x_i))$$

Thus, Maximizing likelihood $\equiv$ Minimizing binary cross entropy.

For general J

let $P_\theta(y=j|x_i) = \dfrac{e^{x^T\theta_j}}{\sum_{k=1}^{J} e^{x^T\theta_k}}$, Softmax

now, likelihood : let $Y \in \mathbb{R}^{N \times J}$, each row corresponds to one of the N samples, and for that sample, that row is a 1-of-J encoding for the class

$$* \; P(Y|X,\theta) = \prod_{i=1}^{N} \prod_{j=1}^{J} P_\theta(y=j|x_i)^{y_{i,j}}$$

$y_{i,j}$ the $[i,j]$ element of Y

$$\log[P(Y|X,\theta)] = \sum_{i=1}^{N}\sum_{j=1}^{J} y_{i,j} \, P_\theta(y=j|x_i)$$

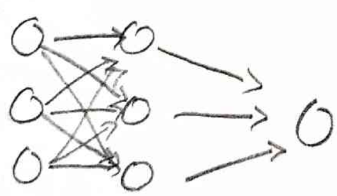
Comments:

Maximizing this likelihood (*) equivalent to minimizing the negative log likelihood $\rightarrow$ which is equal to the cross-entropy cost-function.

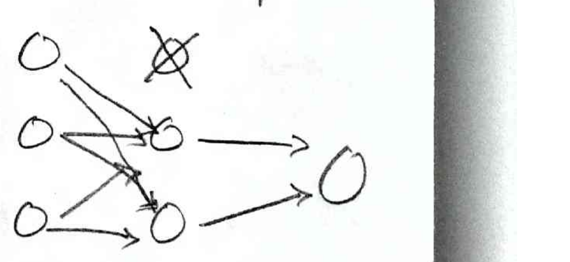
iii)

$$E_\lambda = -\sum_{i=1}^{N}\sum_{j=1}^{J} y_{i,j}\log(b_{i,j}) + \underbrace{\frac{\lambda}{2}\sum_{i=1}^{Z}\sum_{j=1}^{J} v_{i,j}^2}_{\text{output layer weights}} + \underbrace{\frac{\lambda}{2}\sum_{i=1}^{d}\sum_{j=1}^{Z} w_{i,j}^2}_{\text{hidden layer weights}}$$

iv) What is dropout?



Every iteration, randomly select some nodes and $\longrightarrow$ remove them with all their incoming and outgoing connections



with the probability of dropping each neuron as a hyperparameter p.

Why dropout?
During training, neurons develop co-dependencies between each other. These co-adaptations increase the complexity of the model, as the individual explanatory power of 1 neuron is curbed, and 2 or more neurons are used to represent the relationship. By arbitrarily dropping neurons, this prevents the formation of complex co-dependencies on the training data, which prevents overfitting.

v) Overfitting occurs when a learning model is too complex, which can occur when there are too many parameters to tune.

When overfitting, the adjusted parameters tend to get very large. Thus, to prevent overfitting, L2 regularization seeks to penalize the network for getting too large.
L2 regularization modifies the objective function

$$E_\lambda = E + \frac{\lambda}{2}\|W\|_2^2$$

Thus, in trying to minimize E_λ, we must balance fitting the data (E) and fitting it too well ($\frac{\lambda}{2}\|W\|_2^2$)

On the other hand, dropout doesn't modify the objective function. Since overfitting is caused, in part, from having too many parameters to adjust, these parameters can develop complex co-adaptations on the training data. Dropout prevents the formation of complex co-dependencies on the training data by randomly dropping some units during the training process.

vi) $$E_\lambda = -\sum_{i=1}^{N} \sum_{j=1}^{J} y_{i,j} \log(b_{i,j})$$
$$+ \frac{\lambda}{2} \sum_{i=1}^{Z} \sum_{j=1}^{J} V_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{d} \sum_{j=1}^{Z} \omega_{ij}^2$$

Equivalently

$$E_\lambda = -\sum_{i=1}^{N} \overset{\in \mathbb{R}^J}{y_i} \log(\overset{\in \mathbb{R}^J}{b_i}) + \frac{\lambda}{2} \sum_{i=1}^{Z} \overset{\in \mathbb{R}^k}{V_i^T V_i} + \frac{\lambda}{2} \sum_{i=1}^{d} \overset{\in \mathbb{R}^Z}{\omega_i^T \omega_i}$$

$$b_i = \sigma\left(V^T h(x_i) + b_2\right)$$

where $h(x_i) = g(\omega^T x_i + b_1)$

$$\frac{\partial E_\lambda}{\partial V_i} = -\sum_{k=1}^{J} y_k \frac{1}{b_k} \cdot \frac{\partial b_k}{\partial V_i} + \lambda V_i$$

$$= -\sum_{k=1}^{J} y_k \frac{1}{b_k} \left(\sigma'\left(V^T h(x_k) + b_2\right) \bullet h(x_i)\right) + \lambda V_i$$

Update $\Delta V_i = -\gamma \frac{\partial E}{\partial V_i}$

$$\frac{\partial E}{\partial w_i} = -\sum_{k=1}^{J} y_k \frac{1}{b_k} \cdot \frac{\partial b_k}{w_i} + \lambda w_i$$

$$= -\sum_{k=1}^{J} y_k \frac{1}{b_k} \left[O'\left(V^T h(x_k) + b_2\right) \right] \frac{\partial h(x_k)}{\partial w_i} + \lambda w_i$$

$$= -\sum_{k=1}^{J} y_k \frac{1}{b_k} \left[O'\left(V^T h(x_k) + b_2\right) \right] g'\left(W^T x_k + b_1\right) \cdot x_i$$

$$= -\sum_{k=1}^{J} y_k \frac{1}{b_k} \left[O'\left(V^T h(x_k) + b_2\right) \right] \left[g'\left(W^T x_k + b_1\right) \right] x_i + \lambda w_i$$

update: $\Delta w_i = -\gamma \frac{\partial E}{\partial w_i}$

c) Hessian is computed by calculating the second derivatives WRT the parameters.

In general, for $W = \{w_1, w_2\}$

$$H = \nabla\nabla E(w) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 w_2} \\ \frac{\partial^2 E}{\partial w_2 w_1} & \frac{\partial^2 E}{\partial w_2^2} \end{bmatrix}$$

$$H_{V_z} = \begin{bmatrix} \frac{\partial^2 E}{\partial V_{z1}^2} & \cdots & \partial^2 E / \partial V_{z1} V_{zk} \\ \vdots & & \\ \partial^2 E / \partial V_{zk} V_{z1} & & \partial^2 E / \partial V_{zk}^2 \end{bmatrix} \quad z = 1, \ldots, Z$$

$$H_{w_i} = \begin{bmatrix} \partial^2 E / \partial w_{i1}^2 & \cdots & \partial^2 E / \partial w_{i1} w_{iz} \\ \vdots & \ddots & \\ \partial^2 E / \partial w_{iz} w_{i1} & & \partial^2 E / \partial w_{iz}^2 \end{bmatrix} \quad d = 1, \dots d$$

The Hessian matrix could be used in gradient descent:

i.e Newton Raphson

$$w^{(new)} = w^{(old)} - H^{-1} \nabla E(w)$$

This iterative scheme converges much faster that gradient descent → thus, less number of epochs before converging to minima.

ii) $V^{(new)} = V^{(old)} - H^{-1} \nabla E(V)$

$V \in \mathbb{R}^{z \times k} \Rightarrow V_1, \dots, V_{z \cdot k}$ parameters

$H_V \in \mathbb{R}^{(zk) \times (zk)}$

Let's say the cost of computing entry i,j of H_V is C

Computing $H_V \rightarrow C \cdot (zk)^2$,

Computing $H_V^{-1} \rightarrow$ using optimized Coppersmith-Winograd algorithm $\in O(n^{2.373})$

$\Rightarrow$ cost $\left((zk)^{2.373}\right)$

Altogether: $(Zk)^{2.373} + c(Zk)^2$
to compute H_v^{-1}, which is extremely expensive to compute every time you want to iterate

$$V^{(new)} = V^{(old)} - H^{-1} \nabla E(V)$$

using Newton-Raphson.