Deep Learning Principles & Applications

Chapter 5 – Improving the way deep neural networks learn

Sudarsan N.S. Acharya (sudarsan.acharya@manipal.edu)

Topics

1. Why deep neural networks tend to overfit?
2. Loss-based Regularization Approaches
3. Why does loss-based regularization work?
4. Dropout regularization - idea
5. Dropout regularization - practical details
6. Why does dropout regularization work?
7. Why initializing weights is important?
8. Weight-initialization techniques
9. Normalizing activations in a deep neural network
10. Batch normalization - practical details

Why deep neural networks tend to overfit?

Why deep neural networks tend to overfit?

- Deep learning models should do well not only on the train data (low training loss),

Why deep neural networks tend to overfit?

- Deep learning models should do well <u>not</u> only on the train data (low training loss), but also on <u>unseen</u> test data (low test loss).

Why deep neural networks tend to overfit?

- Deep learning models should do well not only on the train data (low training loss), but also on unseen test data (low test loss).
- Deep learning models, by construction, are highly nonlinear due to the presence of multiple neurons and layers with their associated weights and nonlinear activation functions.

Why deep neural networks tend to overfit?

- Deep learning models should do well not only on the train data (low training loss), but also on unseen test data (low test loss).
- Deep learning models, by construction, are highly nonlinear due to the presence of multiple neurons and layers with their associated weights and nonlinear activation functions.
- Complex models like these, when over-trained on the train data start learning the noise in the data and tend to learn from specific features.

Why deep neural networks tend to overfit?

- Deep learning models should do well not only on the train data (low training loss), but also on unseen test data (low test loss).
- Deep learning models, by construction, are highly nonlinear due to the presence of multiple neurons and layers with their associated weights and nonlinear activation functions.
- Complex models like these, when over-trained on the train data start learning the noise in the data and tend to learn from specific features.
- One immediate way to address over-training (a.k.a. overfitting) is to have more training data, say by augmenting the training dataset.

Why deep neural networks tend to overfit?

- Deep learning models should do well not only on the train data (low training loss), but also on unseen test data (low test loss).
- Deep learning models, by construction, are highly nonlinear due to the presence of multiple neurons and layers with their associated weights and nonlinear activation functions.
- Complex models like these, when over-trained on the train data start learning the noise in the data and tend to learn from specific features.
- One immediate way to address over-training (a.k.a. overfitting) is to have more training data, say by augmenting the training dataset.
- However, weights getting disproportionately bigger is the root cause of overfitting;

Why deep neural networks tend to overfit?

- Deep learning models should do well not only on the train data (low training loss), but also on unseen test data (low test loss).
- Deep learning models, by construction, are highly nonlinear due to the presence of multiple neurons and layers with their associated weights and nonlinear activation functions.
- Complex models like these, when over-trained on the train data start learning the noise in the data and tend to learn from specific features.
- One immediate way to address over-training (a.k.a. overfitting) is to have more training data, say by augmenting the training dataset.
- However, weights getting disproportionately bigger is the root cause of overfitting; more nodes/layers, the more likely this is to happen.

Loss-based Regularization Approaches

Loss-based Regularization Approaches

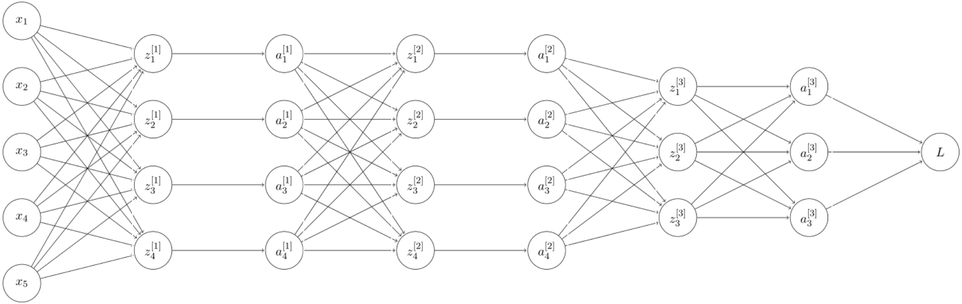
Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values);

Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):

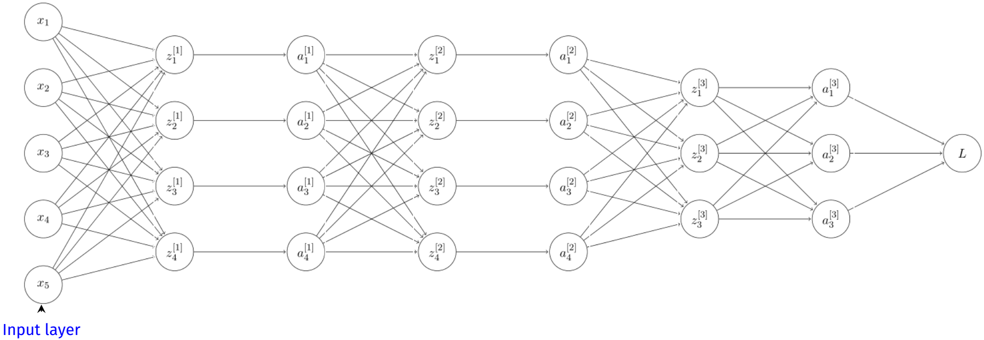
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



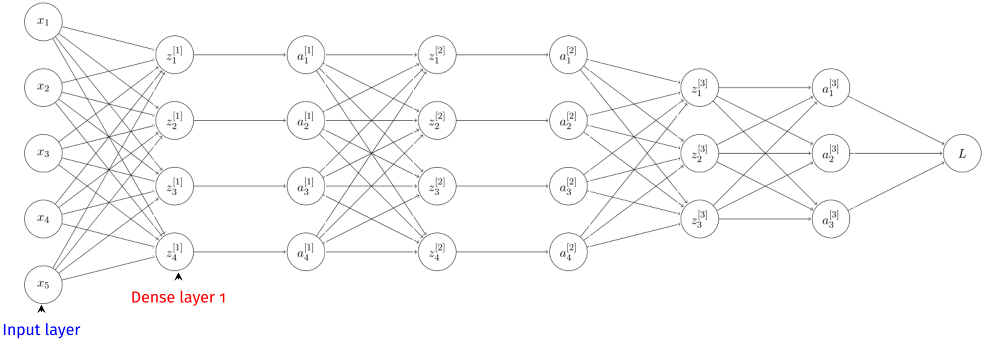
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



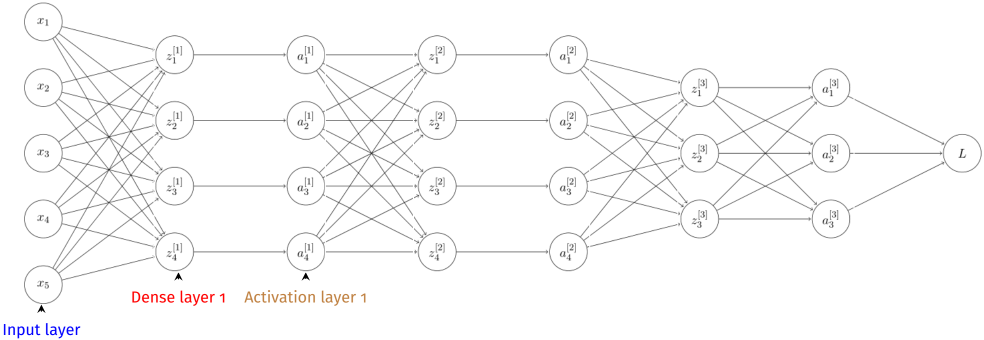
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



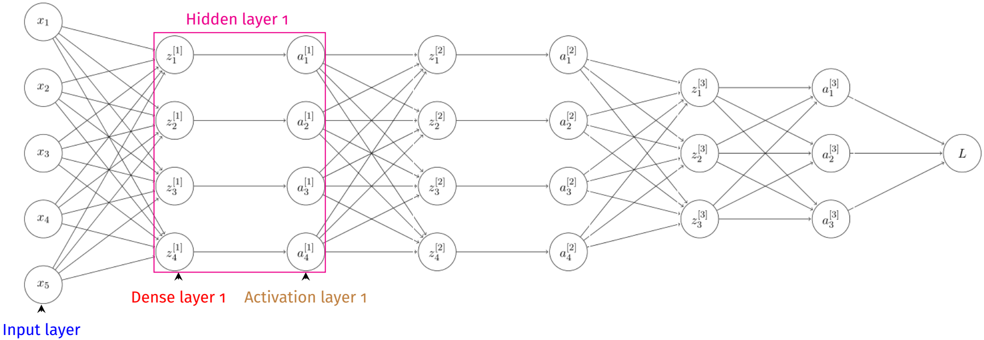
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



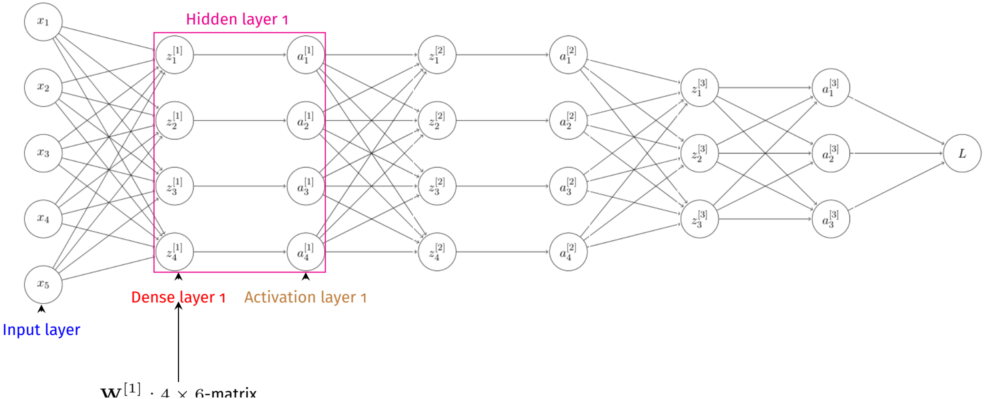
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



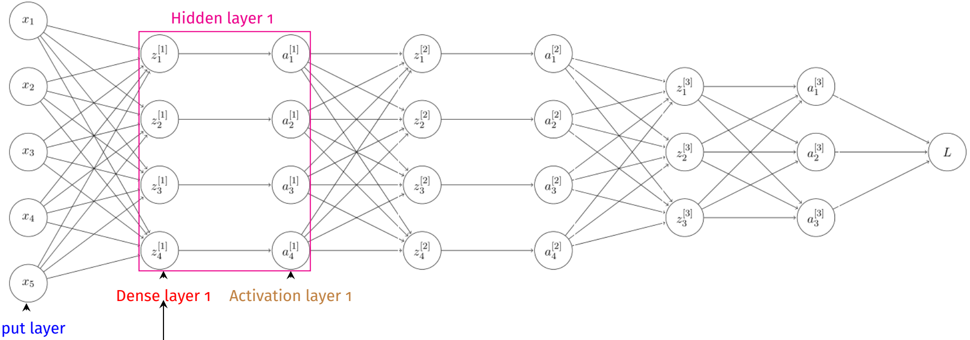
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



Loss-based Regularization Approaches

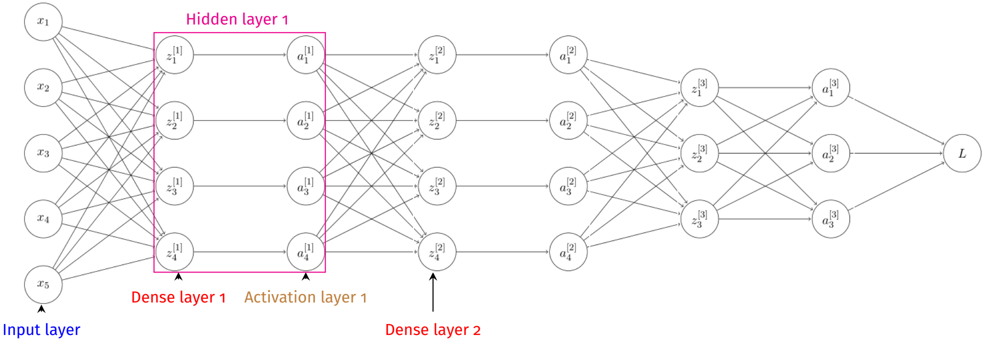
Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



Loss = $L\left(\mathbf{W}^{[1]}\right) + \lambda\left(\|w_{1,1:5}^{[1]}\|^2 + \|w_{2,1:5}^{[1]}\|^2 + \|w_{3,1:5}^{[1]}\|^2 + \|w_{4,1:5}^{[1]}\|^2\right)$

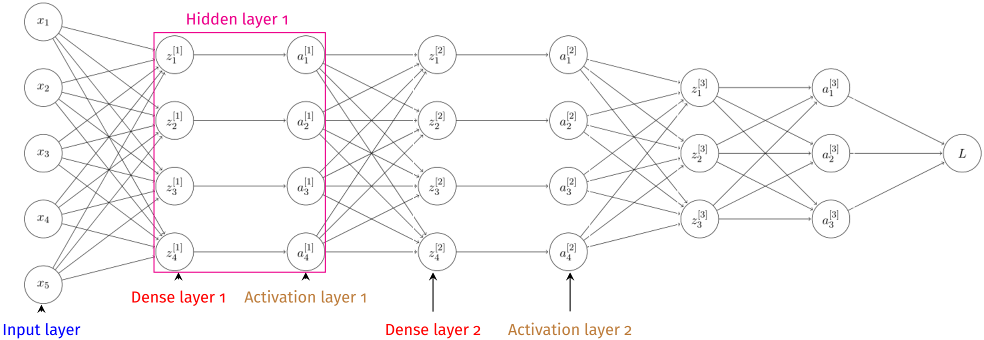
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



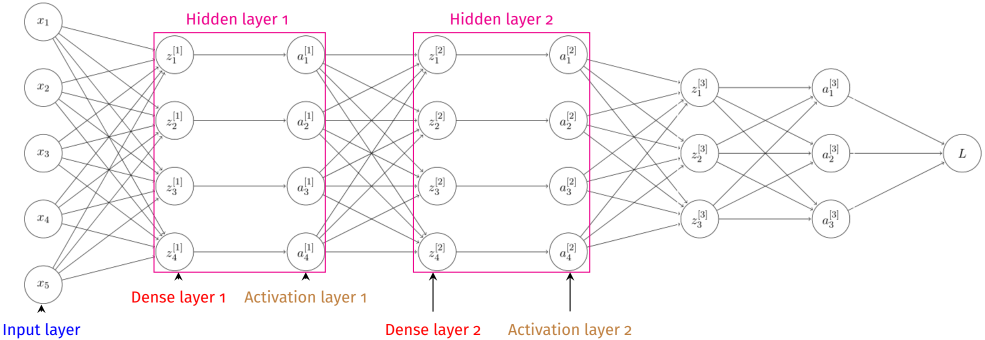
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



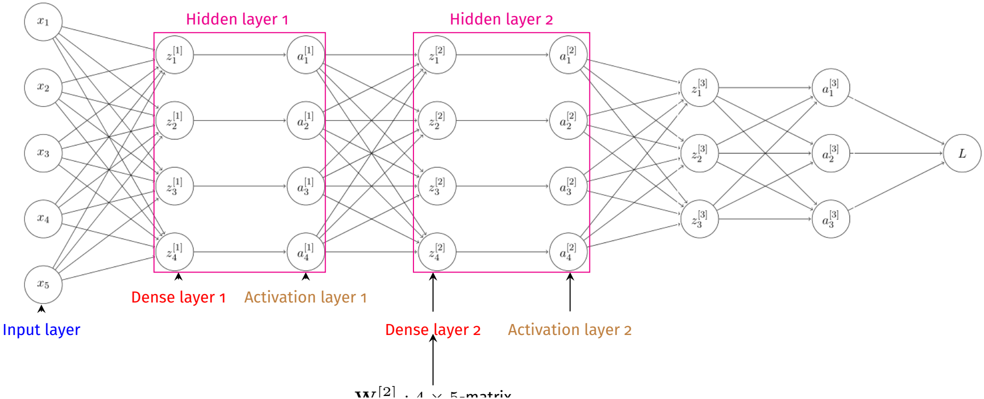
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



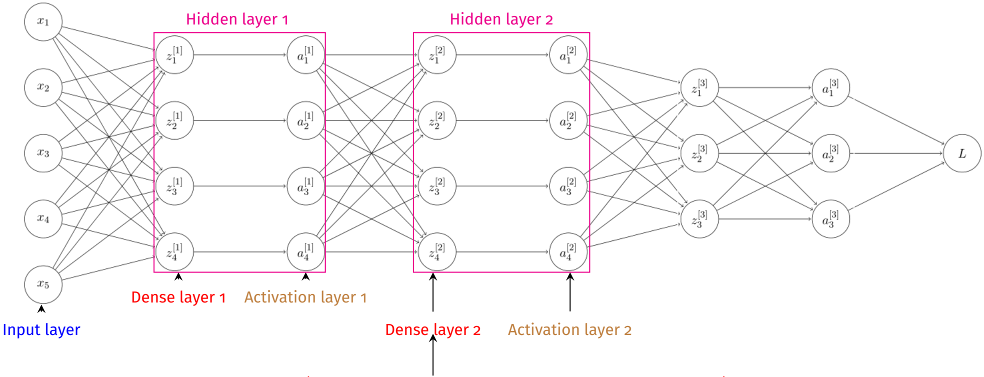
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



Loss-based Regularization Approaches

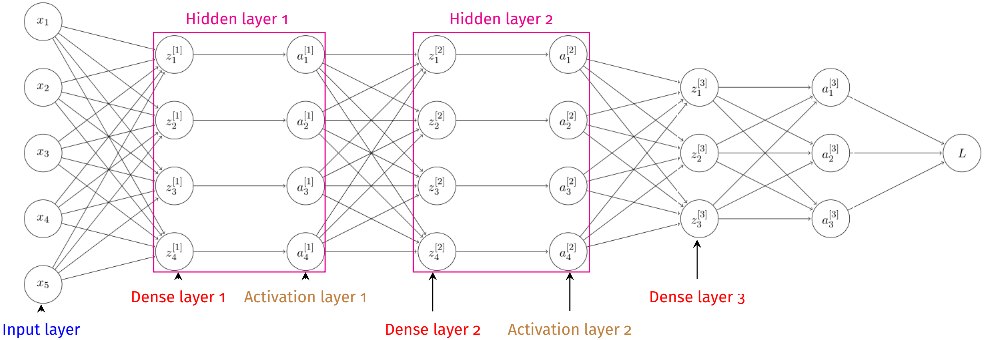
Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



Loss = $L\left(\mathbf{W}^{[2]}\right) + \lambda\left(\|w^{[2]}\|^2 + \|w^{[2]}\|^2 + \|w^{[2]}\|^2 + \|w^{[2]}\|^2\right)$

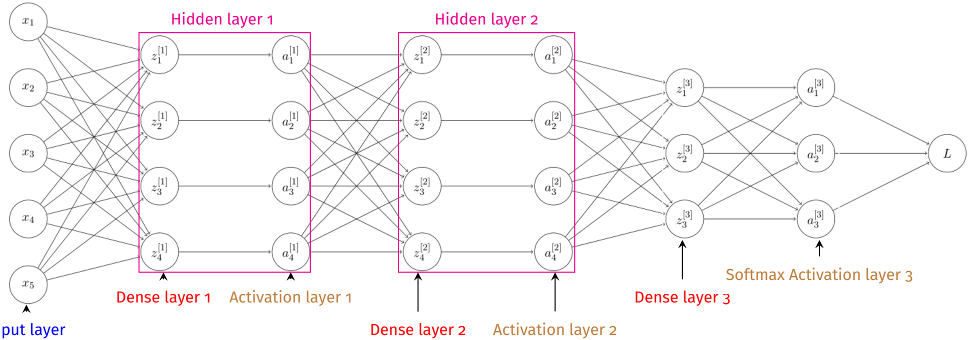
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



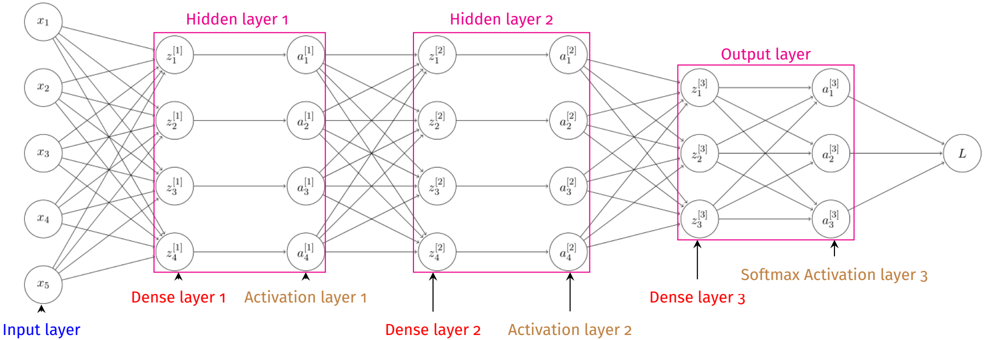
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



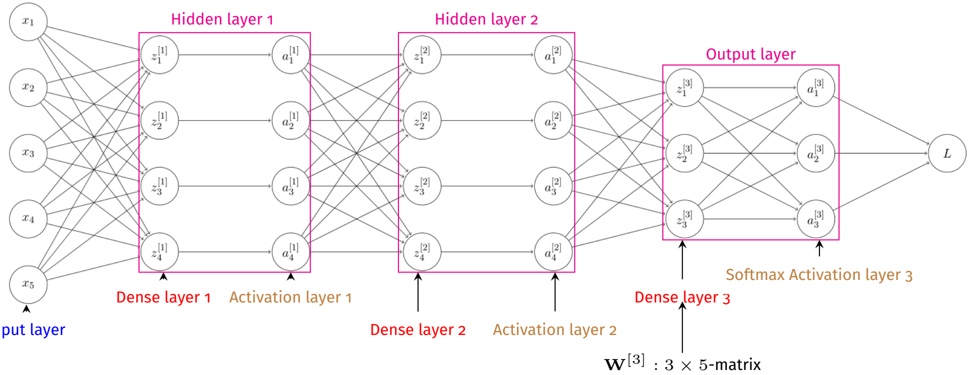
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



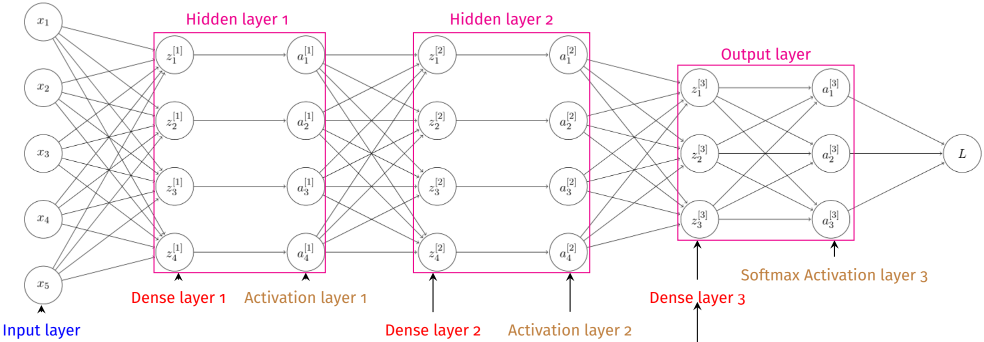
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



Loss-based Regularization Approaches

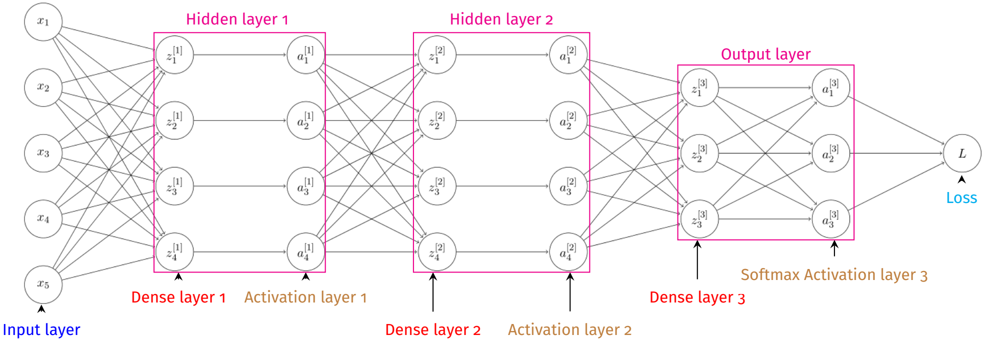
Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):



Loss = $L\left(\mathbf{W}^{[3]}\right) + \lambda\left(\left\|w_{1,1:4}^{[3]}\right\|^2 + \left\|w_{2,1:4}^{[3]}\right\|^2 + \left\|w_{3,1:4}^{[3]}\right\|^2\right)$

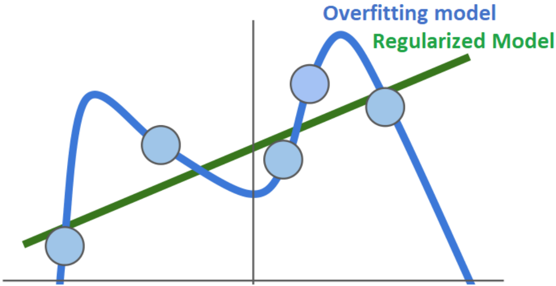
Loss-based Regularization Approaches

Recall loss-based regularization which is achieved by adding to the average training data loss constraints on the weights (not to the bias values); for a deep neural network, regularization is applied to the weights owned by each dense layer (example below shows L_2 regularization with bias):

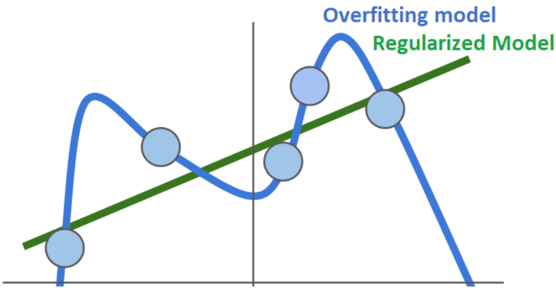


Why does loss-based regularization work?

Why does loss-based regularization work?

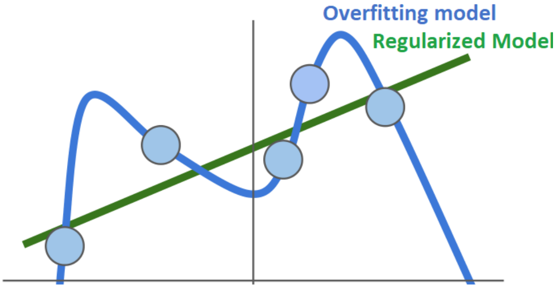


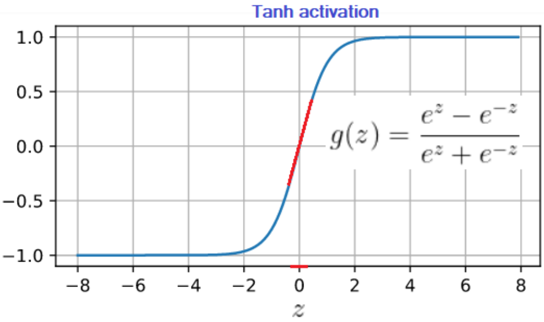
Why does loss-based regularization work?



Regularization shrinks the weights uniformly close to zero thus keeping the model simple.

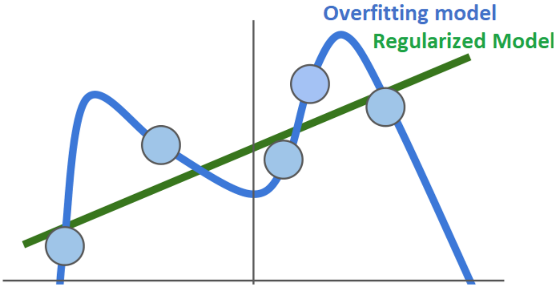
Why does loss-based regularization work?



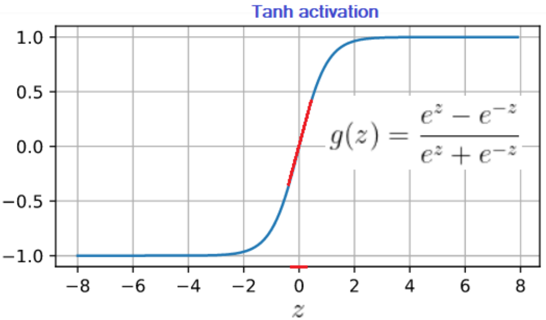


Regularization shrinks the weights uniformly close to zero thus keeping the model simple.

Why does loss-based regularization work?

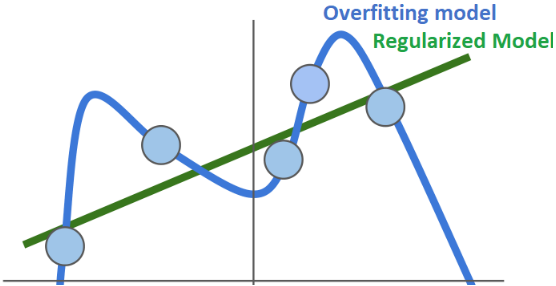


Regularization shrinks the weights uniformly close to zero thus keeping the model simple.

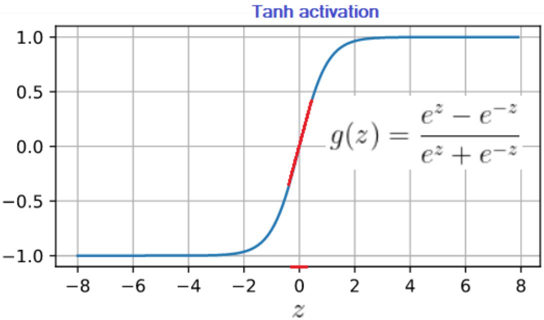


Recall $\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]}$:

Why does loss-based regularization work?

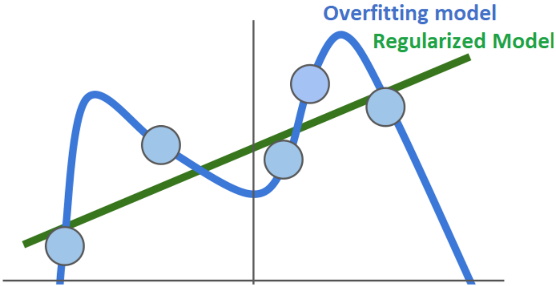


Regularization shrinks the weights uniformly close to zero thus keeping the model simple.

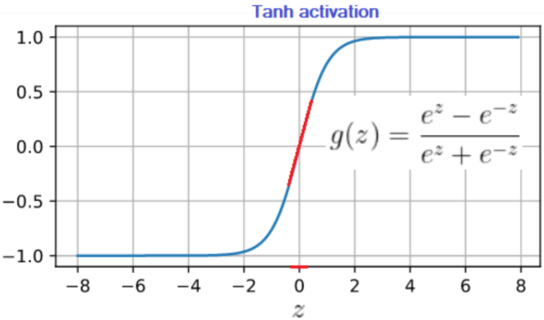


Recall $\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]}$: small weights keep the raw scores small

Why does loss-based regularization work?



Regularization shrinks the weights uniformly close to zero thus keeping the model simple.



Recall $\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]}$: small weights keep the raw scores small thus keeping the activations in the linear zone, thus keeping the model

Dropout regularization - idea

Dropout regularization - idea

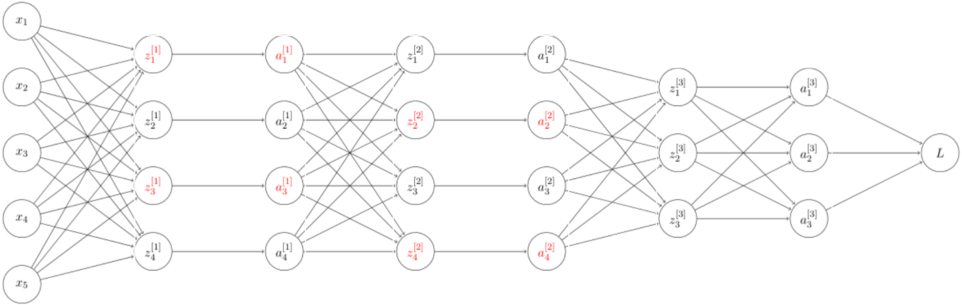
Another approach for regularization is as follows:

Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:

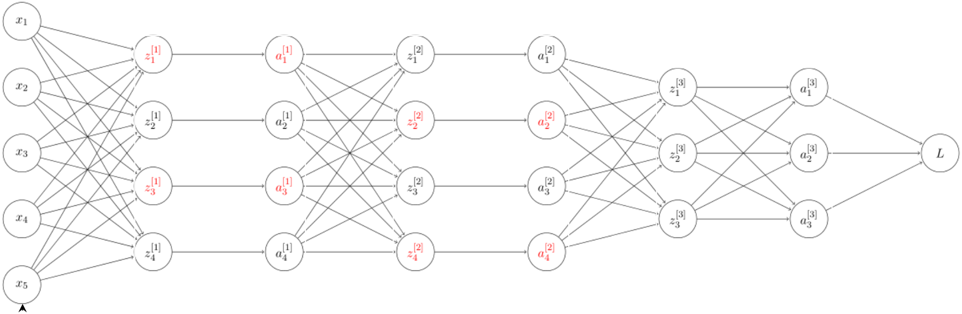
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



Dropout regularization - idea

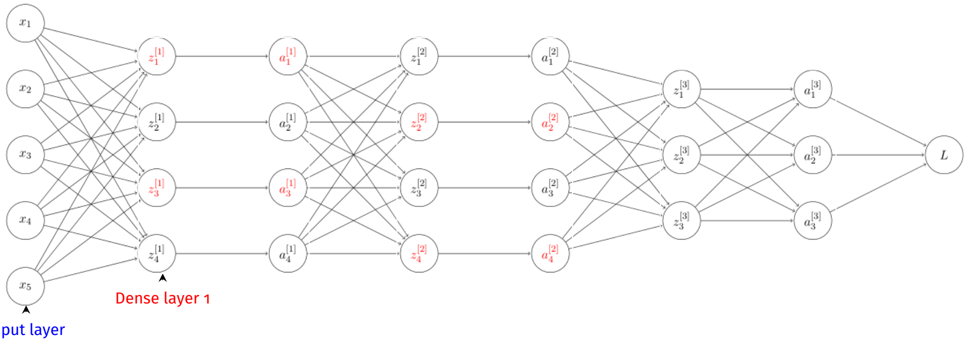
Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



Input layer

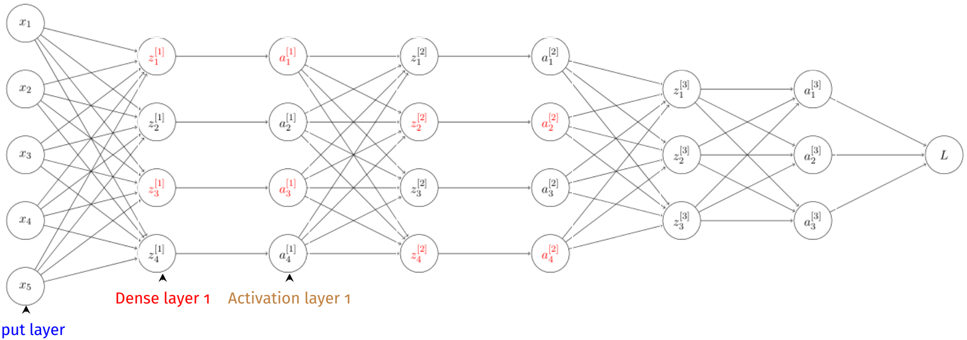
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



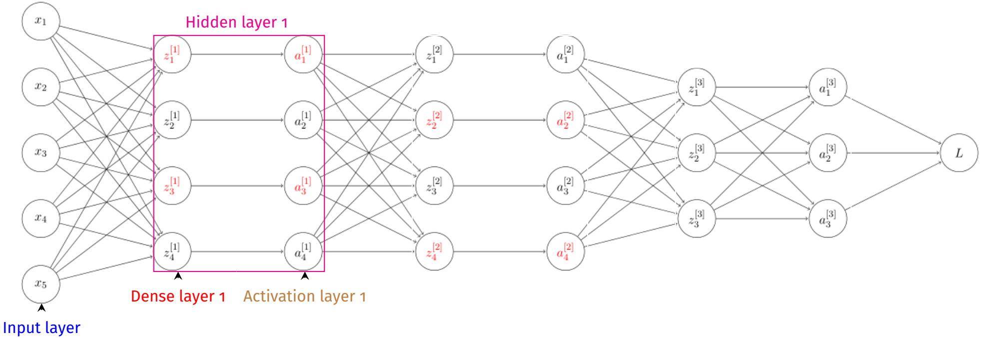
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



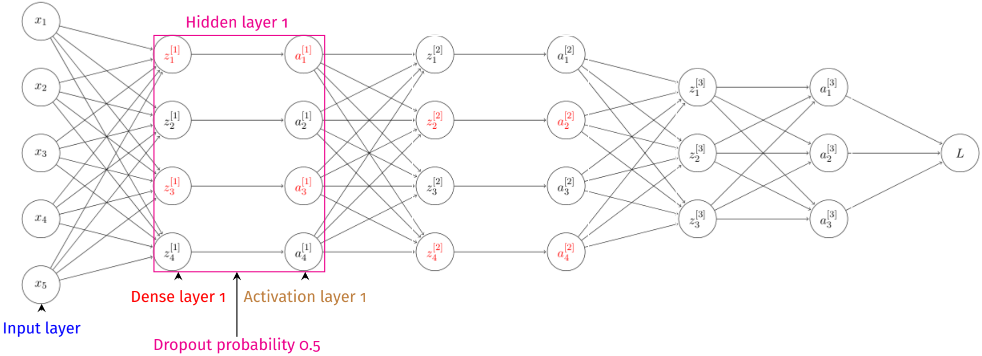
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



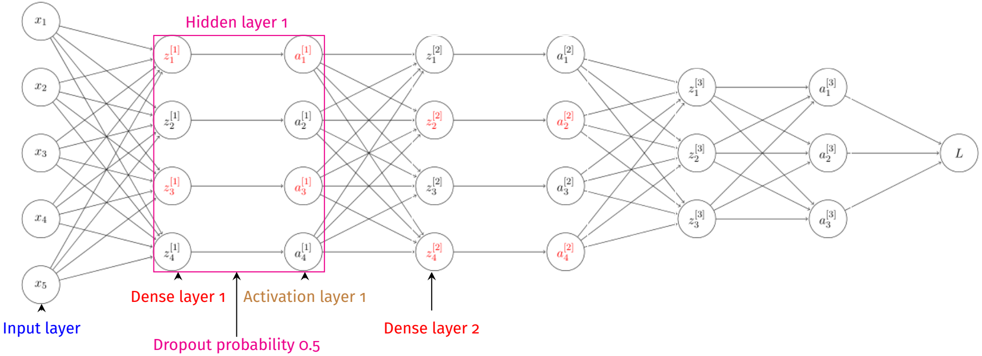
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



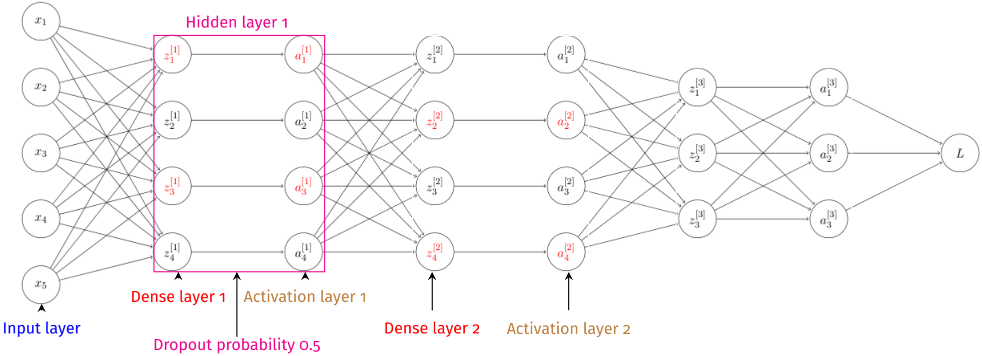
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



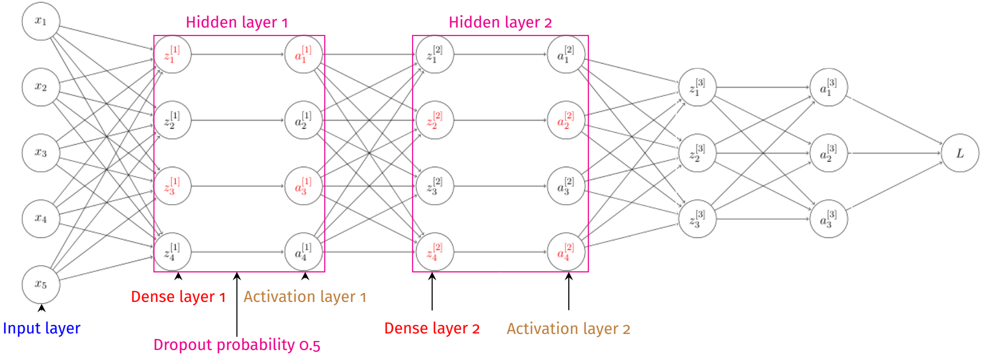
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



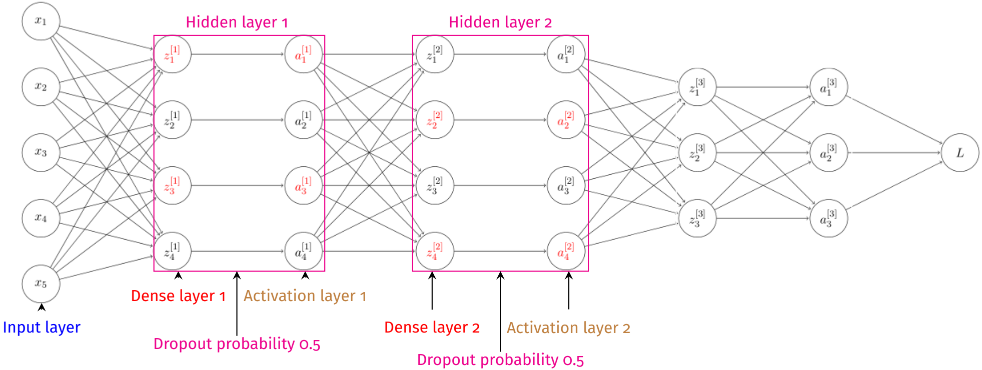
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



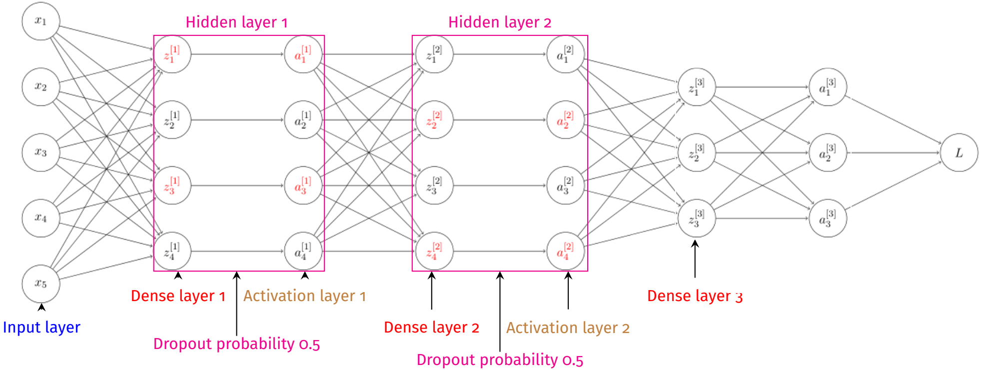
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



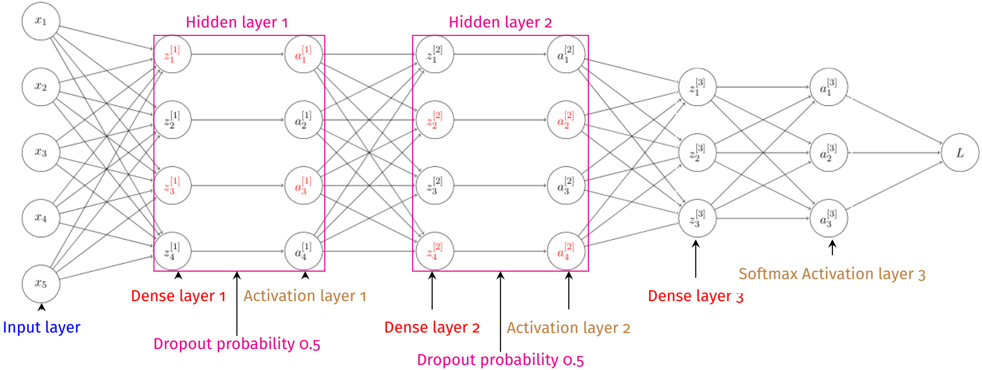
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



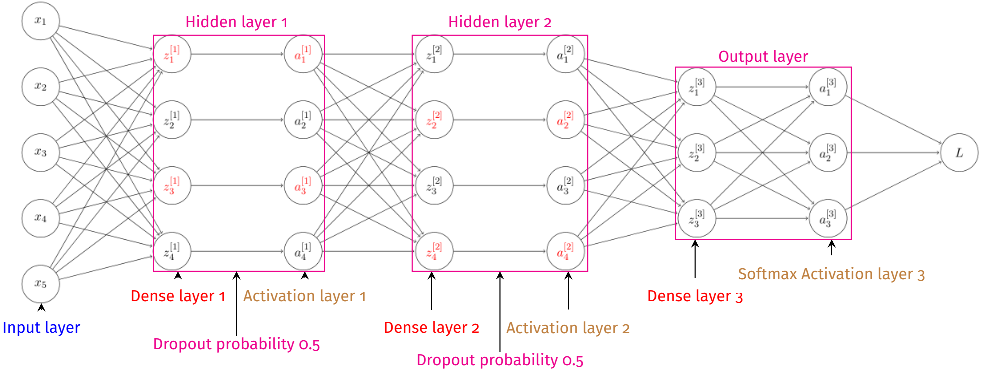
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



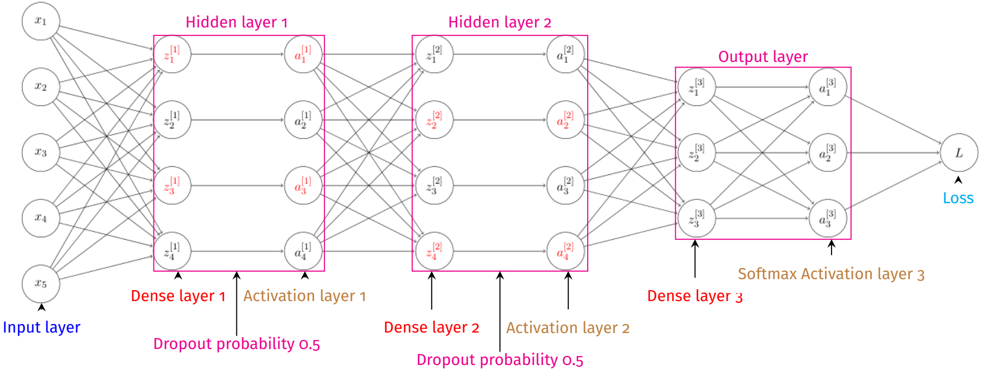
Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



Dropout regularization - idea

Another approach for regularization is as follows: during training using a particular batch of samples, in each hidden layer (dense + activation), nodes are randomly eliminated:



Dropout regularization - idea continued

Dropout regularization - idea continued

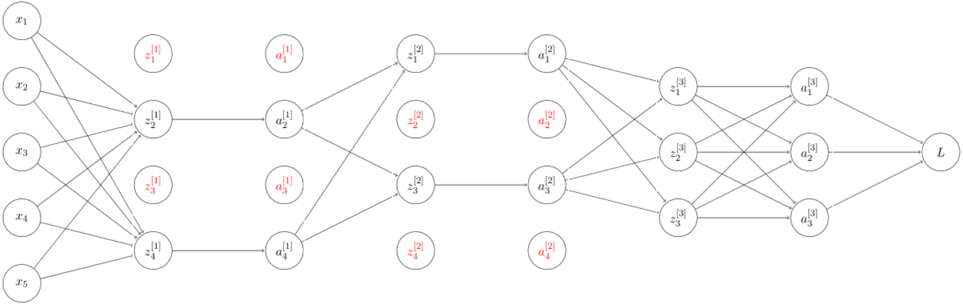
The dropped out nodes do not contribute to the training process (forward and backward propagation):

Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):

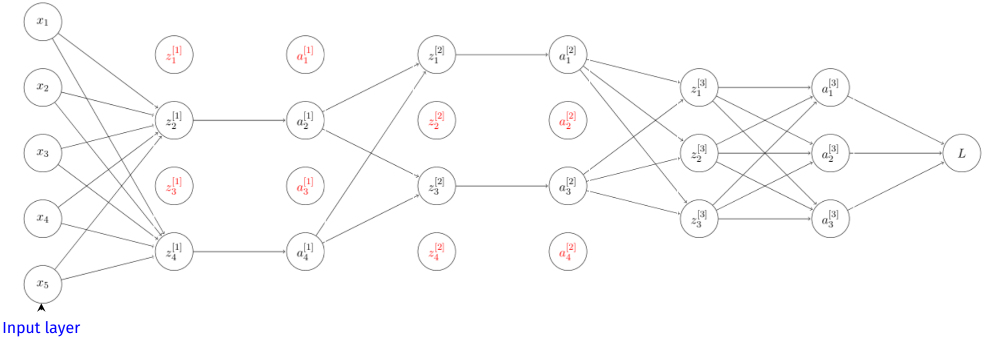
Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



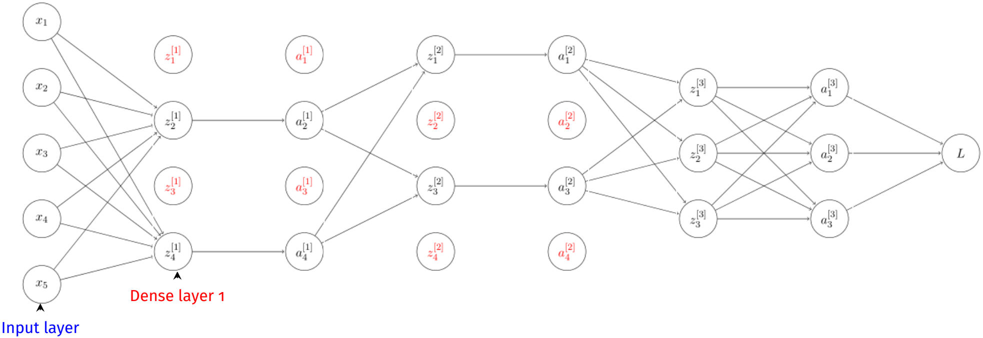
Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



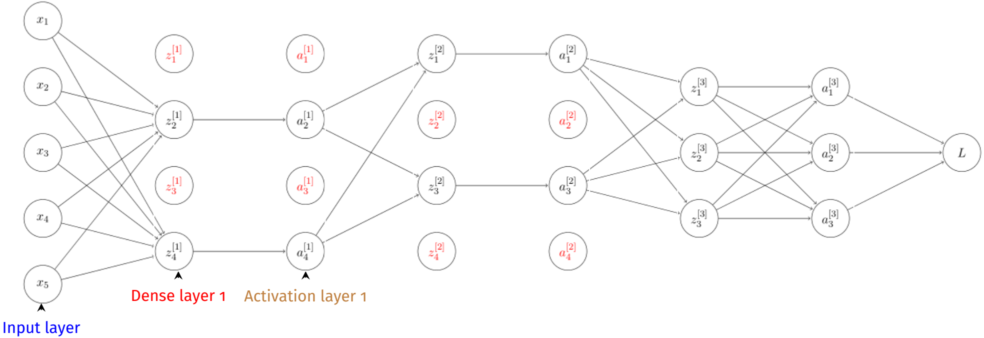
Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



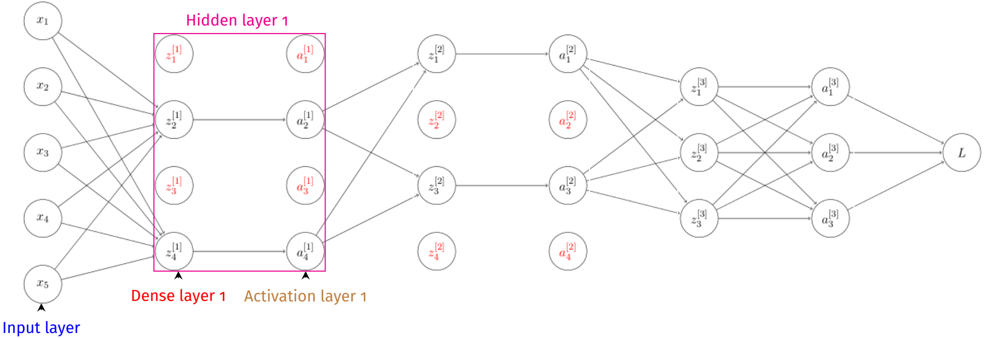
Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



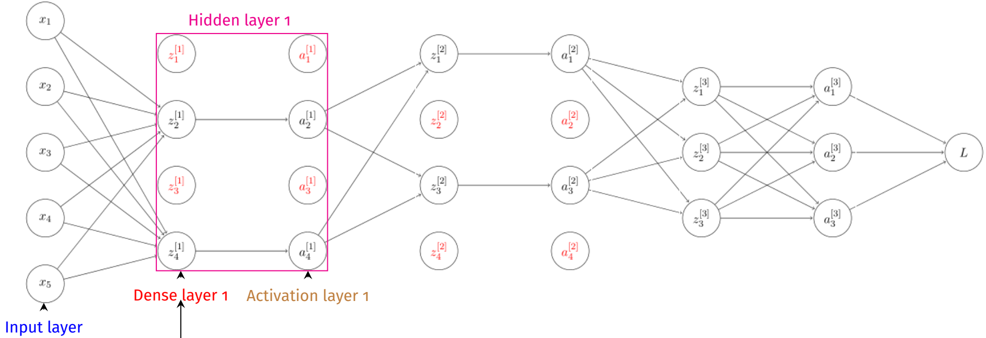
Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



Dropout regularization - idea continued

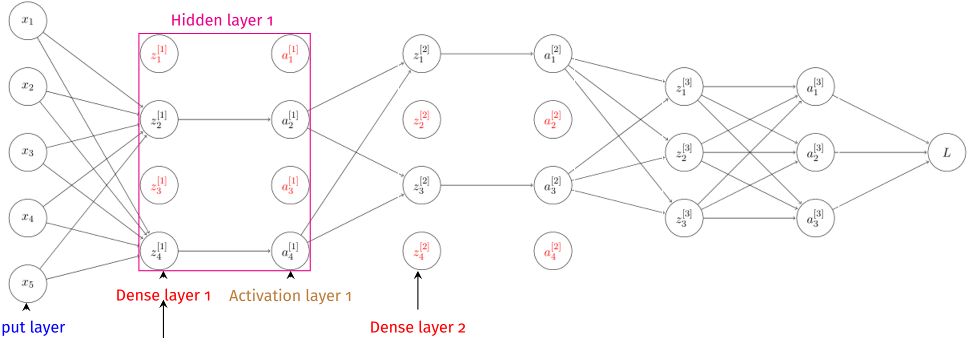
The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



weights matrix $\mathbf{W}^{[1]}$: 4×6-matrix$\rightarrow$ 2×6-matrix

Dropout regularization - idea continued

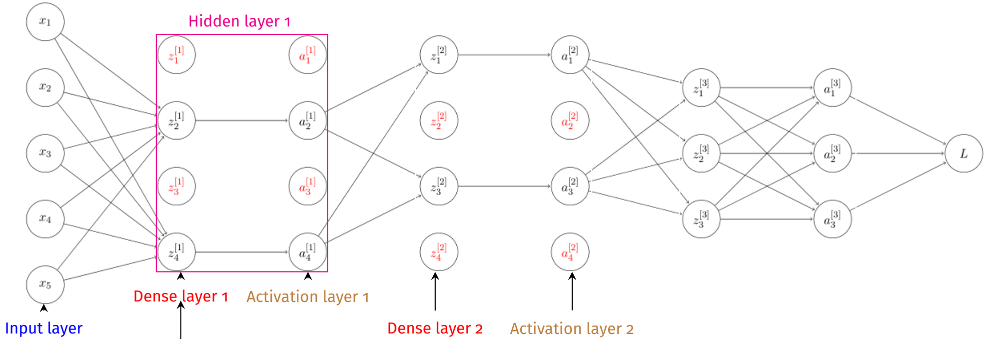
The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



weights matrix $\mathbf{W}^{[1]}$: 4×6-matrix$\rightarrow$ 2×6-matrix

Dropout regularization - idea continued

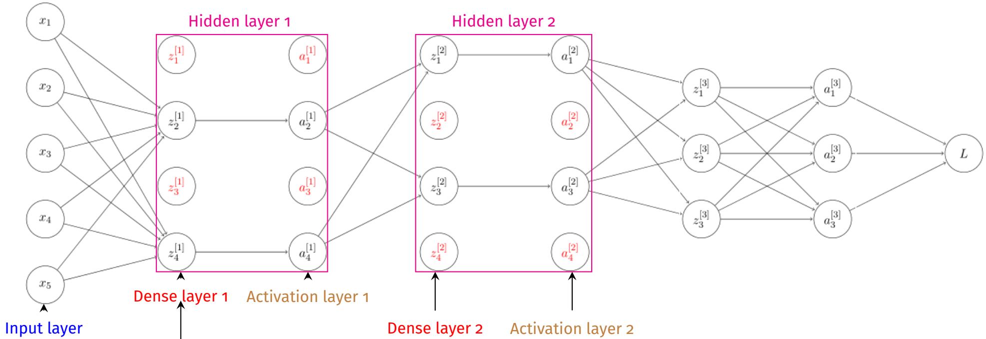
The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



weights matrix $\mathbf{W}^{[1]}$: 4×6-matrix$\rightarrow$ 2×6-matrix

Dropout regularization - idea continued

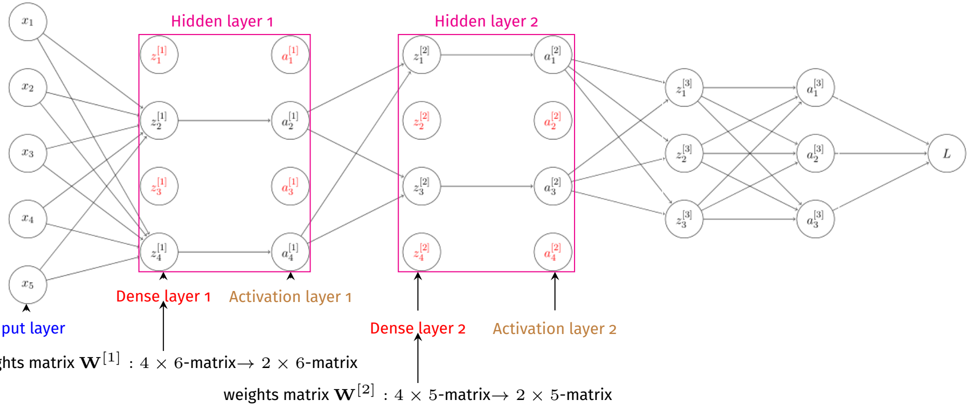
The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



weights matrix $\mathbf{W}^{[1]}$: 4×6-matrix$\rightarrow$ 2×6-matrix

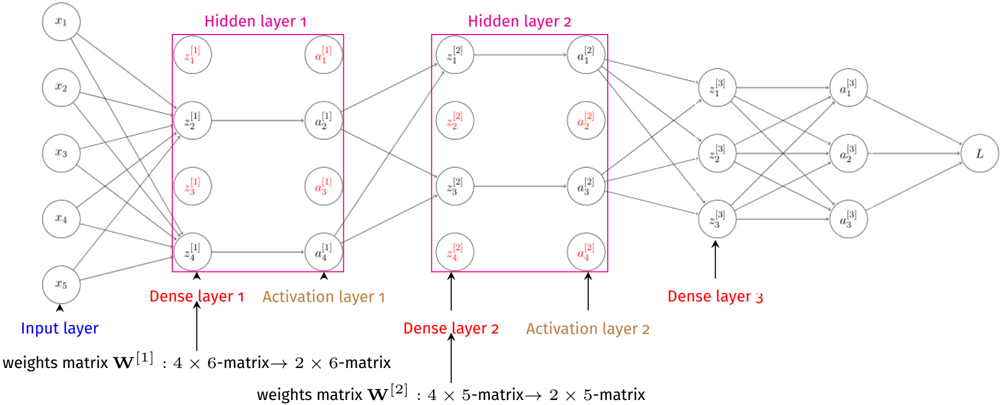
Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



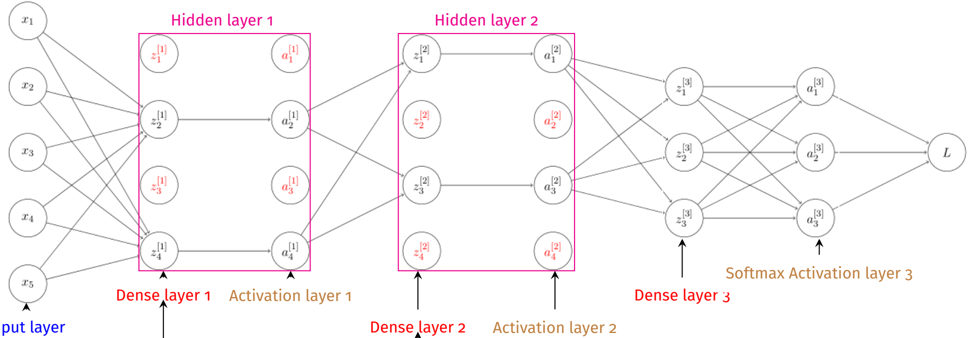
Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):

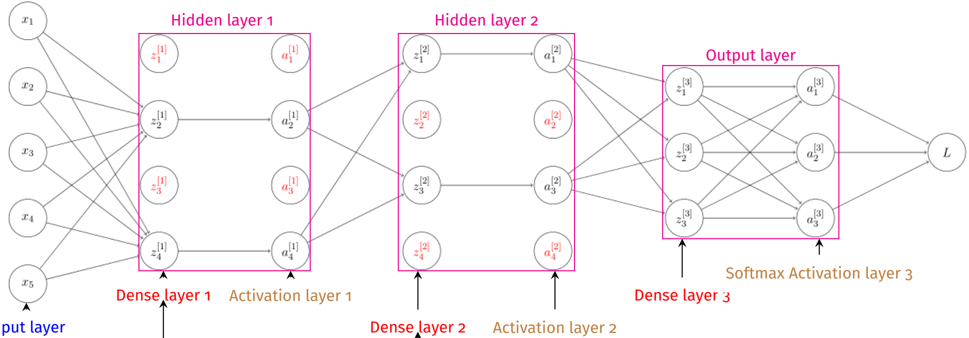


weights matrix $\mathbf{W}^{[1]}$: 4×6-matrix$\rightarrow$ 2×6-matrix

weights matrix $\mathbf{W}^{[2]}$: 4×5-matrix$\rightarrow$ 2×5-matrix

Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):

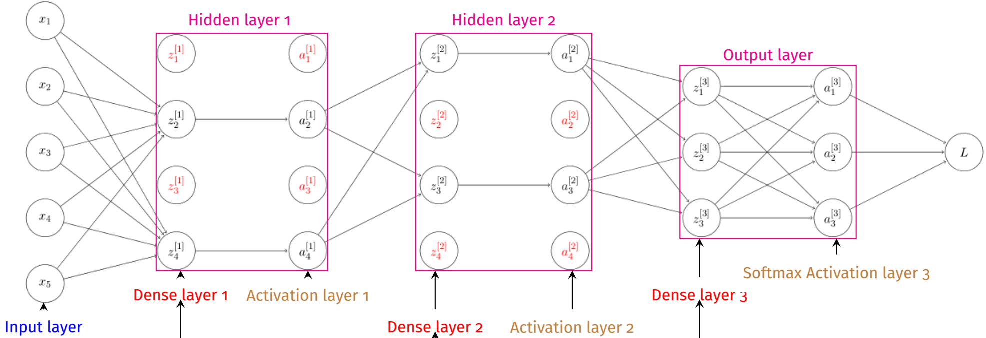


weights matrix $\mathbf{W}^{[1]}$: 4×6-matrix$\rightarrow$ 2×6-matrix

weights matrix $\mathbf{W}^{[2]}$: 4×5-matrix$\rightarrow$ 2×5-matrix

Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



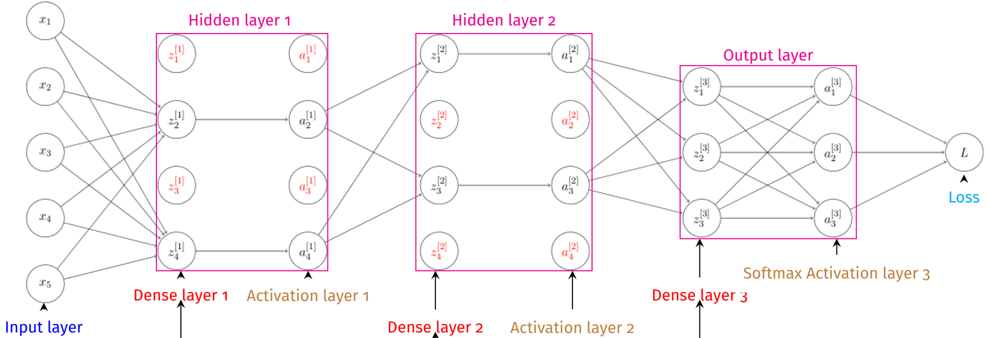
weights matrix $\mathbf{W}^{[1]}$: 4×6-matrix$\rightarrow$ 2×6-matrix

weights matrix $\mathbf{W}^{[2]}$: 3×5-matrix$\rightarrow$ 3×3-matrix

weights matrix $\mathbf{W}^{[2]}$: 4×5-matrix$\rightarrow$ 2×5-matrix

Dropout regularization - idea continued

The dropped out nodes do not contribute to the training process (forward and backward propagation): note the apparent change in the shape of the associated dense layer's weight matrices (bias feature included):



weights matrix $\mathbf{W}^{[1]}$: 4×6-matrix$\rightarrow$ 2×6-matrix

weights matrix $\mathbf{W}^{[2]}$: 3×5-matrix$\rightarrow$ 3×3-matrix

weights matrix $\mathbf{W}^{[2]}$: 4×5-matrix$\rightarrow$ 2×5-matrix

Dropout regularization - practical details

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b:

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b: recall that the shape of $\mathbf{A}^{[l]}$ is $n^{[l]} \times b$, where b is the batch size..

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b: recall that the shape of $\mathbf{A}^{[l]}$ is $n^{[l]} \times b$, where b is the batch size..
- Generate a random matrix $\mathbf{D}^{[l]}$ of the same shape as $\mathbf{A}^{[l]}$ such that its elements are uniform random numbers from 0 through 1.

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b: recall that the shape of $\mathbf{A}^{[l]}$ is $n^{[l]} \times b$, where b is the batch size..
- Generate a random matrix $\mathbf{D}^{[l]}$ of the same shape as $\mathbf{A}^{[l]}$ such that its elements are uniform random numbers from 0 through 1.
- Using a probability p of retaining a node in layer l (which is one minus the probability of dropout),

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b: recall that the shape of $\mathbf{A}^{[l]}$ is $n^{[l]} \times b$, where b is the batch size..
- Generate a random matrix $\mathbf{D}^{[l]}$ of the same shape as $\mathbf{A}^{[l]}$ such that its elements are uniform random numbers from 0 through 1.
- Using a probability p of retaining a node in layer l (which is one minus the probability of dropout), reset the elements of $\mathbf{D}^{[l]}$ as $\mathbf{D}^{[l]} \leq p$ which results in a matrix full of zeros and ones.

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b: recall that the shape of $\mathbf{A}^{[l]}$ is $n^{[l]} \times b$, where b is the batch size..
- Generate a random matrix $\mathbf{D}^{[l]}$ of the same shape as $\mathbf{A}^{[l]}$ such that its elements are uniform random numbers from 0 through 1.
- Using a probability p of retaining a node in layer l (which is one minus the probability of dropout), reset the elements of $\mathbf{D}^{[l]}$ as $\mathbf{D}^{[l]} \leq p$ which results in a matrix full of zeros and ones.
- During forward propagation, replace $\mathbf{A}^{[l]}$ by $\mathbf{A}^{[l]} \odot \mathbf{D}^{[l]}$ (Hadamard product) thus zeroing out $(1-p)\%$ of elements in each column of $\mathbf{A}^{[l]}$.

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b: recall that the shape of $\mathbf{A}^{[l]}$ is $n^{[l]} \times b$, where b is the batch size..
- Generate a random matrix $\mathbf{D}^{[l]}$ of the same shape as $\mathbf{A}^{[l]}$ such that its elements are uniform random numbers from 0 through 1.
- Using a probability p of retaining a node in layer l (which is one minus the probability of dropout), reset the elements of $\mathbf{D}^{[l]}$ as $\mathbf{D}^{[l]} \leq p$ which results in a matrix full of zeros and ones.
- During forward propagation, replace $\mathbf{A}^{[l]}$ by $\mathbf{A}^{[l]} \odot \mathbf{D}^{[l]}$ (Hadamard product) thus zeroing out $(1-p)\%$ of elements in each column of $\mathbf{A}^{[l]}$.
- Replace $\mathbf{A}^{[l]}$ by $\mathbf{A}^{[l]}/p$

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b: recall that the shape of $\mathbf{A}^{[l]}$ is $n^{[l]} \times b$, where b is the batch size..
- Generate a random matrix $\mathbf{D}^{[l]}$ of the same shape as $\mathbf{A}^{[l]}$ such that its elements are uniform random numbers from 0 through 1.
- Using a probability p of retaining a node in layer l (which is one minus the probability of dropout), reset the elements of $\mathbf{D}^{[l]}$ as $\mathbf{D}^{[l]} \leq p$ which results in a matrix full of zeros and ones.
- During forward propagation, replace $\mathbf{A}^{[l]}$ by $\mathbf{A}^{[l]} \odot \mathbf{D}^{[l]}$ (Hadamard product) thus zeroing out $(1-p)\%$ of elements in each column of $\mathbf{A}^{[l]}$.
- Replace $\mathbf{A}^{[l]}$ by $\mathbf{A}^{[l]}/p$ so that the next layers raw scores $\mathbf{Z}^{[l+1]} = \mathbf{W}^{[l+1]}\mathbf{A}^{[l]}$ do not go down in expected value because of dropping nodes.

Dropout regularization - practical details

- Inverted dropout: a popular dropout regularization technique.
- Consider layer index l in a deep neural network associated with the activated scores matrix $\mathbf{A}^{[l]}$ for a training batch of size b: recall that the shape of $\mathbf{A}^{[l]}$ is $n^{[l]} \times b$, where b is the batch size..
- Generate a random matrix $\mathbf{D}^{[l]}$ of the same shape as $\mathbf{A}^{[l]}$ such that its elements are uniform random numbers from 0 through 1.
- Using a probability p of retaining a node in layer l (which is one minus the probability of dropout), reset the elements of $\mathbf{D}^{[l]}$ as $\mathbf{D}^{[l]} \leq p$ which results in a matrix full of zeros and ones.
- During forward propagation, replace $\mathbf{A}^{[l]}$ by $\mathbf{A}^{[l]} \odot \mathbf{D}^{[l]}$ (Hadamard product) thus zeroing out $(1-p)\%$ of elements in each column of $\mathbf{A}^{[l]}$.
- Replace $\mathbf{A}^{[l]}$ by $\mathbf{A}^{[l]}/p$ so that the next layers raw scores $\mathbf{Z}^{[l+1]} = \mathbf{W}^{[l+1]}\mathbf{A}^{[l]}$ do not go down in expected value because of dropping nodes.
- Most important: dropout is not applied at test time.

Why does dropout regularization work?

Why does dropout regularization work?

- Dropout ensures that a neuron in a particular layer learns its weights by not relying on a particular set of features because different nodes (read features) in the previous layer are dropped out randomly for each sample in a training batch.

Why does dropout regularization work?

- Dropout ensures that a neuron in a particular layer learns its weights by not relying on a particular set of features because different nodes (read features) in the previous layer are dropped out randomly for each sample in a training batch.
- This has an effect similar to that of shrinking weights towards zero by L_2 regularization.

Why does dropout regularization work?

- Dropout ensures that a neuron in a particular layer learns its weights by not relying on a particular set of features because different nodes (read features) in the previous layer are dropped out randomly for each sample in a training batch.
- This has an effect similar to that of shrinking weights towards zero by L_2 regularization.
- In a deep neural network with multiple layers, layers with higher number of nodes are typically associated with a higher dropout probability that ones with smaller number of nodes.

Why initializing weights is important?

Why initializing weights is important?

- Recall that the training loss in a deep neural network is a function of the weights associated with the dense layers.

Why initializing weights is important?

- Recall that the training loss in a deep neural network is a function of the weights associated with the dense layers.
- Initializing those weights appropriately can have a significant impact on how fast the learning happens.

Why initializing weights is important?

- Recall that the training loss in a deep neural network is a function of the weights associated with the dense layers.
- Initializing those weights appropriately can have a significant impact on how fast the learning happens.
- Initializing the same weights for all neurons in a particular dense layer will result in symmetric learning and will prevent the neurons from learning different things.

Why initializing weights is important?

- Recall that the training loss in a deep neural network is a function of the weights associated with the dense layers.
- Initializing those weights appropriately can have a significant impact on how fast the learning happens.
- Initializing the same weights for all neurons in a particular dense layer will result in symmetric learning and will prevent the neurons from learning different things.
- Initializing the weights too small may lead to slow learning.

Why initializing weights is important?

- Recall that the training loss in a deep neural network is a function of the weights associated with the dense layers.
- Initializing those weights appropriately can have a significant impact on how fast the learning happens.
- Initializing the same weights for all neurons in a particular dense layer will result in symmetric learning and will prevent the neurons from learning different things.
- Initializing the weights too small may lead to slow learning.
- Initializing the weights too big may lead to divergence.

Why initializing weights is important?

- Recall that the training loss in a deep neural network is a function of the weights associated with the dense layers.
- Initializing those weights appropriately can have a significant impact on how fast the learning happens.
- Initializing the same weights for all neurons in a particular dense layer will result in symmetric learning and will prevent the neurons from learning different things.
- Initializing the weights too small may lead to slow learning.
- Initializing the weights too big may lead to divergence.
- An excellent resource to visualize this: https://www.deeplearning.ai/ai-notes/initialization/index.html

Why initializing weights is important? – continued

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right)$

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right)$

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$,

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = \mathbf{W}^{[l]^{\mathrm{T}}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.
- This means, weights initialized with small values $\Rightarrow$

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.
- This means, weights initialized with small values $\Rightarrow$ vanishing gradients,

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.
- This means, weights initialized with small values $\Rightarrow$ vanishing gradients, weights initialized with large values $\Rightarrow$

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.
- This means, weights initialized with small values $\Rightarrow$ vanishing gradients, weights initialized with large values $\Rightarrow$ exploding gradients.

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.
- This means, weights initialized with small values $\Rightarrow$ vanishing gradients, weights initialized with large values $\Rightarrow$ exploding gradients.
- A good initialization of weights should result in:

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = \mathbf{W}^{[l]^{\mathrm{T}}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.
- This means, weights initialized with small values $\Rightarrow$ vanishing gradients, weights initialized with large values $\Rightarrow$ exploding gradients.
- A good initialization of weights should result in: (1) zero mean of the activations

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = {\mathbf{W}^{[l]}}^{\mathrm{T}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.
- This means, weights initialized with small values $\Rightarrow$ vanishing gradients, weights initialized with large values $\Rightarrow$ exploding gradients.
- A good initialization of weights should result in: (1) zero mean of the activations (2) constant variance of activations across the layers,

Why initializing weights is important? – continued

- For a single sample, the local gradient of dense layer l (output w.r.t. its input) is $\nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{z}^{[l]}\right) = \nabla_{\mathbf{a}^{[l-1]}}\left(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]}\right) = \mathbf{W}^{[l]^{\mathrm{T}}}$.
- For a single sample, the local gradient of activation layer $l-1$ (the previous layer) is $\nabla_{\mathbf{z}^{[l-1]}}\left(\mathbf{a}^{[l-1]}\right)$, which for ReLU activation is a diagonal matrix whose entries are $I\left(\mathbf{z}^{[l-1]} > 0\right)$.
- Terms such as the above get progressively multiplied during backward propagation for calculating gradients of loss w.r.t. weights.
- This means, weights initialized with small values $\Rightarrow$ vanishing gradients, weights initialized with large values $\Rightarrow$ exploding gradients.
- A good initialization of weights should result in: (1) zero mean of the activations (2) constant variance of activations across the layers, which guarantess that there will be no vanishing or exploding gradients.

Weight-initialization techniques

Weight-initialization techniques

-

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.
-

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.
- Xavier initialization for tanh activation:

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.
- Xavier initialization for tanh activation: initialize $\underbrace{w^{[l]}_{:,\,1:n^{[l-1]}-1} \sim N\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]} + n^{[l]}}\right)}_{\text{weights}}$

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.
- Xavier initialization for tanh activation: initialize $\underbrace{w^{[l]}_{:,\,1:n^{[l-1]}-1} \sim N\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]} + n^{[l]}}\right)}_{\text{weights}}$ and $\underbrace{w^{[l]}_{:,\,n^{[l-1]}} = 0}_{\text{bias}}$.

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.
- Xavier initialization for tanh activation: initialize $\underbrace{w^{[l]}_{:,\,1:n^{[l-1]}-1} \sim N\left(\mu = 0, \sigma^2 = \dfrac{2}{n^{[l-1]} + n^{[l]}}\right)}_{\text{weights}}$ and $\underbrace{w^{[l]}_{:,\,n^{[l-1]}} = 0}_{\text{bias}}$.
-

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.
- Xavier initialization for tanh activation: initialize $\underbrace{w^{[l]}_{:,\,1:n^{[l-1]}-1} \sim N\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]} + n^{[l]}}\right)}_{\text{weights}}$ and $\underbrace{w^{[l]}_{:,\,n^{[l-1]}} = 0}_{\text{bias}}$.
- He initialization for ReLU activation:

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.
- Xavier initialization for tanh activation: initialize $\underbrace{w^{[l]}_{:,\,1:n^{[l-1]}-1} \sim N\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]} + n^{[l]}}\right)}_{\text{weights}}$ and $\underbrace{w^{[l]}_{:,\,n^{[l-1]}} = 0}_{\text{bias}}$.
- He initialization for ReLU activation: initialize $\underbrace{w^{[l]}_{:,\,1:n^{[l-1]}-1} \sim N\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]}}\right)}_{\text{weights}}$

Weight-initialization techniques

- For dense layer l, the weights matrix $\mathbf{W}^{[l]}$ has shape $n^{[l]} \times n^{[l-1]}$.
- Xavier initialization for tanh activation: initialize $\underbrace{w^{[l]}_{:,\,1:n^{[l-1]}-1} \sim N\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]} + n^{[l]}}\right)}_{\text{weights}}$ and $\underbrace{w^{[l]}_{:,\,n^{[l-1]}}}_{\text{bias}} = 0$.
- He initialization for ReLU activation: initialize $\underbrace{w^{[l]}_{:,\,1:n^{[l-1]}-1} \sim N\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]}}\right)}_{\text{weights}}$ and $\underbrace{w^{[l]}_{:,\,n^{[l-1]}}}_{\text{bias}} = 0$.

Normalizing activations in a deep neural network

Normalizing activations in a deep neural network

- Standardization of an input feature (such as heart rate, for example) is the process of subtracting the mean of that feature across all samples and dividing by standard deviation of that feature.

Normalizing activations in a deep neural network

- Standardization of an input feature (such as heart rate, for example) is the process of subtracting the mean of that feature across all samples and dividing by standard deviation of that feature.
- Standardization results in positive and negative values for the feature (above and below average) that are independent of units used to measure that feature (division by standard deviation).

Normalizing activations in a deep neural network

- Standardization of an input feature (such as heart rate, for example) is the process of subtracting the mean of that feature across all samples and dividing by standard deviation of that feature.
- Standardization results in positive and negative values for the feature (above and below average) that are independent of units used to measure that feature (division by standard deviation).
- This typically speeds up the learning of the weights associated with the next dense layer.

Normalizing activations in a deep neural network

- Standardization of an input feature (such as heart rate, for example) is the process of subtracting the mean of that feature across all samples and dividing by standard deviation of that feature.
- Standardization results in positive and negative values for the feature (above and below average) that are independent of units used to measure that feature (division by standard deviation).
- This typically speeds up the learning of the weights associated with the next dense layer.
- In the deep learning context, standardization is referred to as normalization.

Normalizing activations in a deep neural network

- Standardization of an input feature (such as heart rate, for example) is the process of subtracting the mean of that feature across all samples and dividing by standard deviation of that feature.
- Standardization results in positive and negative values for the feature (above and below average) that are independent of units used to measure that feature (division by standard deviation).
- This typically speeds up the learning of the weights associated with the next dense layer.
- In the deep learning context, standardization is referred to as normalization.
- The same idea can be applied to speed up the learning of the weights associated with deeper dense layers in a deep neural network.

Batch normalization - practical details

Batch normalization - practical details

- Consider the hidden layer in layer l of a deep neural network.

Batch normalization - practical details

- Consider the hidden layer in layer l of a deep neural network.
- Weights to be learned from a batch of samples of size is b.

Batch normalization - practical details

- Consider the hidden layer in layer l of a deep neural network.
- Weights to be learned from a batch of samples of size is b.
- The raw scores matrix $\mathbf{Z}^{[l]} = \begin{bmatrix} \mathbf{z}^{[l](1)} & \mathbf{z}^{[l](2)} & \ldots & \mathbf{z}^{[l](b)} \end{bmatrix}$.

Batch normalization - practical details

- Consider the hidden layer in layer l of a deep neural network.
- Weights to be learned from a batch of samples of size is b.
- The raw scores matrix $\mathbf{Z}^{[l]} = \begin{bmatrix} \mathbf{z}^{[l](1)} & \mathbf{z}^{[l](2)} & \dots & \mathbf{z}^{[l](b)} \end{bmatrix}$.
- Calculate mean raw scores in the batch $\boldsymbol{\mu} = \frac{1}{b}\sum_{i=1}^{b} \mathbf{z}^{[l](i)}$.

Batch normalization - practical details

- Consider the hidden layer in layer l of a deep neural network.
- Weights to be learned from a batch of samples of size is b.
- The raw scores matrix $\mathbf{Z}^{[l]} = \begin{bmatrix} \mathbf{z}^{[l](1)} & \mathbf{z}^{[l](2)} & \dots & \mathbf{z}^{[l](b)} \end{bmatrix}$.
- Calculate mean raw scores in the batch $\boldsymbol{\mu} = \frac{1}{b}\sum_{i=1}^{b} \mathbf{z}^{[l](i)}$.
- Calculate standard deviation of raw scores $\boldsymbol{\sigma}^2 = \frac{1}{b}\sum_{i=1}^{b} \left(\mathbf{z}^{[l](i)} - \boldsymbol{\mu}\right)^2$.

Batch normalization - practical details

- Consider the hidden layer in layer l of a deep neural network.
- Weights to be learned from a batch of samples of size is b.
- The raw scores matrix $\mathbf{Z}^{[l]} = \begin{bmatrix} \mathbf{z}^{[l](1)} & \mathbf{z}^{[l](2)} & \dots & \mathbf{z}^{[l](b)} \end{bmatrix}$.
- Calculate mean raw scores in the batch $\boldsymbol{\mu} = \frac{1}{b}\sum_{i=1}^{b} \mathbf{z}^{[l](i)}$.
- Calculate standard deviation of raw scores $\boldsymbol{\sigma}^2 = \frac{1}{b}\sum_{i=1}^{b} \left(\mathbf{z}^{[l](i)} - \boldsymbol{\mu}\right)^2$.
- Calculate standardized raw scores for each sample in the batch as $\mathbf{z}_{\text{norm}}^{[l](i)} = \frac{\mathbf{z}^{[l](i)} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}}$, where the nonzero constant ϵ is for numerical stability.

Batch normalization - practical details

- Consider the hidden layer in layer l of a deep neural network.
- Weights to be learned from a batch of samples of size is b.
- The raw scores matrix $\mathbf{Z}^{[l]} = \begin{bmatrix} \mathbf{z}^{[l](1)} & \mathbf{z}^{[l](2)} & \ldots & \mathbf{z}^{[l](b)} \end{bmatrix}$.
- Calculate mean raw scores in the batch $\boldsymbol{\mu} = \frac{1}{b}\sum_{i=1}^{b} \mathbf{z}^{[l](i)}$.
- Calculate standard deviation of raw scores $\boldsymbol{\sigma}^2 = \frac{1}{b}\sum_{i=1}^{b} \left(\mathbf{z}^{[l](i)} - \boldsymbol{\mu}\right)^2$.
- Calculate standardized raw scores for each sample in the batch as $\mathbf{z}_{\text{norm}}^{[l](i)} = \frac{\mathbf{z}^{[l](i)} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}}$, where the nonzero constant ϵ is for numerical stability.
- Calculate batch norm-updated raw scores for each sample in the batch as $\tilde{\mathbf{z}}^{[l](i)} = \gamma \mathbf{z}_{\text{norm}}^{[l](i)} + \beta$,

Batch normalization - practical details

- Consider the hidden layer in layer l of a deep neural network.
- Weights to be learned from a batch of samples of size is b.
- The raw scores matrix $\mathbf{Z}^{[l]} = \begin{bmatrix} \mathbf{z}^{[l](1)} & \mathbf{z}^{[l](2)} & \dots & \mathbf{z}^{[l](b)} \end{bmatrix}$.
- Calculate mean raw scores in the batch $\boldsymbol{\mu} = \frac{1}{b} \sum_{i=1}^{b} \mathbf{z}^{[l](i)}$.
- Calculate standard deviation of raw scores $\boldsymbol{\sigma}^2 = \frac{1}{b} \sum_{i=1}^{b} \left(\mathbf{z}^{[l](i)} - \boldsymbol{\mu} \right)^2$.
- Calculate standardized raw scores for each sample in the batch as $\mathbf{z}_{\text{norm}}^{[l](i)} = \frac{\mathbf{z}^{[l](i)} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}}$, where the nonzero constant ϵ is for numerical stability.
- Calculate batch norm-updated raw scores for each sample in the batch as $\tilde{\mathbf{z}}^{[l](i)} = \gamma \mathbf{z}_{\text{norm}}^{[l](i)} + \beta$, where γ are β are learnable parameters just like the weights which let the mean and variance of the batch norm-updated raw scores to not to be tied to 0 and 1, respectively, because of the normalization step