Deep Neural Network

AIML Module 5

Seetha Parameswaran

BITS Pilani

The author of this deck, Prof. Seetha Parameswaran, is gratefully acknowledging the authors who made their course materials freely available online.

Regularization Techniques

What we Learn….

4.1 Model Selection

4.2 Underfitting, and Overfitting

4.3 L1 and L2 Regularization

4.4 Dropout

4.5 Challenges - Vanishing and Exploding Gradients, Covariance shift

4.6 Parameter Initialization

4.7 Batch Normalization

Generalization in DNN

Generalization

- Goal is to discover patterns that generalize.
  - The goal is to discover patterns that capture regularities in the underlying population from which our training set was drawn.
  - Models are trained a sample of data.
  - When working with finite samples, we run the risk that we might discover apparent associations that turn out not to hold up when we collect more data or on newer samples.
- The trained model should predict for newer or unseen data. This problem is called generalization.

Training Error and Generalization Error

- **Training error** is the error of our model as calculated on the training dataset.
  - Obtained while training the model.
- **Generalization error** is the expectation of our model's error, if an infinite stream of additional data examples drawn from the same underlying data distribution as the original sample were applied on the model.
  - Cannot be computed, but estimated.
  - Estimate the generalization error by applying the model to an independent test set, constituted of a random selection of data examples that were withheld from the training set.

Model Complexity

- Simple models and abundant data
  - Expect the generalization error to resemble the training error.
- More complex models and fewer examples
  - Expect the training error to go down but the generalization gap to grow.
- Model complexity
  - A model with more parameters might be considered more complex.
  - A model whose parameters can take a wider range of values might be more complex.
  - A neural network model that takes more training iterations are more complex, and one subject to early stopping (fewer training iterations) are less complex.
-

Factors that influence the generalizability of a model

1. The number of tunable parameters.
   - When the number of tunable parameters, called the degrees of freedom, is large, models tend to be more susceptible to overfitting.
2. The values taken by the parameters.
   - When weights can take a wider range of values, models can be more susceptible to overfitting.
3. The number of training examples.
   - It is trivially easy to overfit a dataset containing only one or two examples even if your model is simple. But overfitting a dataset with millions of examples requires an extremely flexible model.
4.

Model Selection

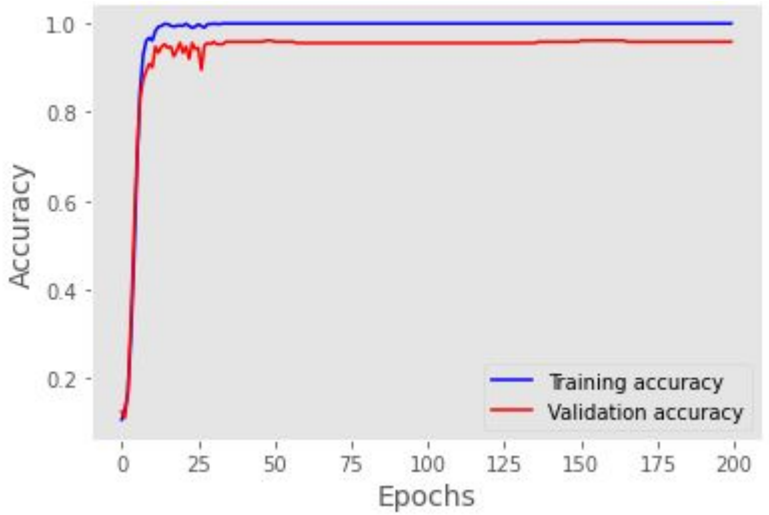
- Model selection is the process of selecting the final model after evaluating several candidate models.
- With MLPs, compare models with
  - different numbers of hidden layers,
  - different numbers of hidden units
  - different activation functions applied to each hidden layer.
- Use Validation dataset to determine the best among our candidate models.

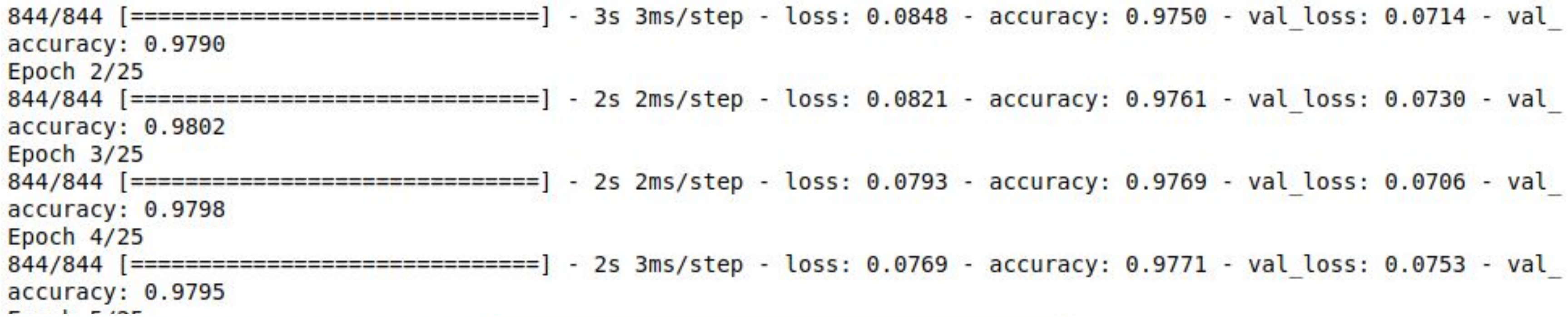
Validation dataset

- Never rely on the test data for model selection.
  - Risk of overfit the test data
- Do not rely solely on the training data for model selection
  - We cannot estimate the generalization error on the very data that we use to train the model.
- Split the data three ways, incorporating a validation dataset (or validation set) in addition to the training and test datasets.
- In deep learning, with millions of data available, the split is generally
  - Training = 98-99 % of the original dataset
  - Validation = 1-2 % of training dataset
  - Testing = 1-2 % of the original dataset

Just Right Model

- High Training accuracy
- High Validation accuracy
- Low Bias and Low Variance
- Usually care more about the validation error than about the gap between the training and validation errors.

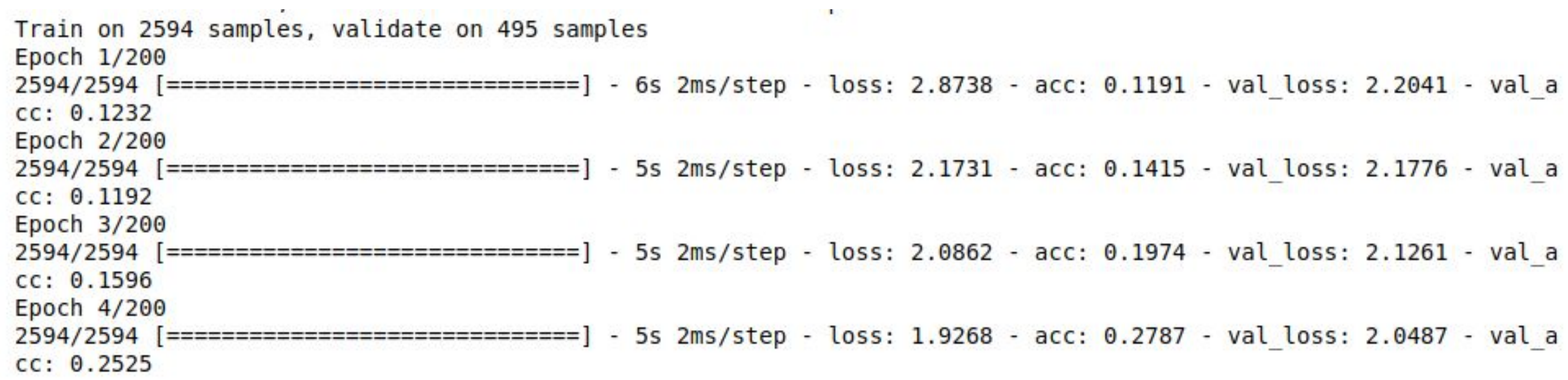




```
844/844 [==============================] - 3s 3ms/step - loss: 0.0848 - accuracy: 0.9750 - val_loss: 0.0714 - val_
accuracy: 0.9790
Epoch 2/25
844/844 [==============================] - 2s 2ms/step - loss: 0.0821 - accuracy: 0.9761 - val_loss: 0.0730 - val_
accuracy: 0.9802
Epoch 3/25
844/844 [==============================] - 2s 2ms/step - loss: 0.0793 - accuracy: 0.9769 - val_loss: 0.0706 - val_
accuracy: 0.9798
Epoch 4/25
844/844 [==============================] - 2s 3ms/step - loss: 0.0769 - accuracy: 0.9771 - val_loss: 0.0753 - val_
accuracy: 0.9795
```

Underfitting

- Low Training accuracy and Low Validation accuracy.



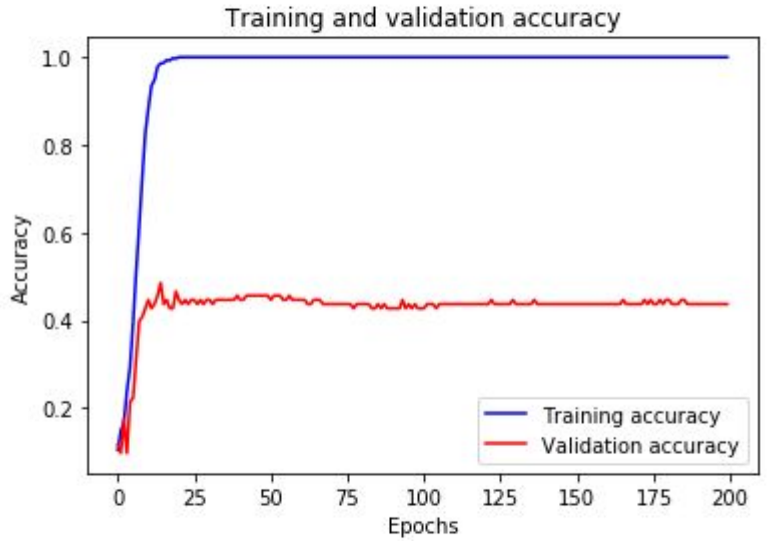
```
Train on 2594 samples, validate on 495 samples
Epoch 1/200
2594/2594 [==============================] - 6s 2ms/step - loss: 2.8738 - acc: 0.1191 - val_loss: 2.2041 - val_a
cc: 0.1232
Epoch 2/200
2594/2594 [==============================] - 5s 2ms/step - loss: 2.1731 - acc: 0.1415 - val_loss: 2.1776 - val_a
cc: 0.1192
Epoch 3/200
2594/2594 [==============================] - 5s 2ms/step - loss: 2.0862 - acc: 0.1974 - val_loss: 2.1261 - val_a
cc: 0.1596
Epoch 4/200
2594/2594 [==============================] - 5s 2ms/step - loss: 1.9268 - acc: 0.2787 - val_loss: 2.0487 - val_a
cc: 0.2525
```

- Training error and validation error are both substantial but there is a little gap between them.
- The model is too simple (insufficiently expressive) to capture the pattern that we are trying to model.
- If generalization gap between our training and validation errors is small, a more complex model may be better.

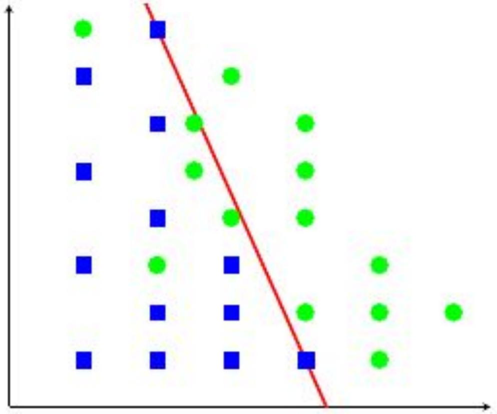
Overfitting

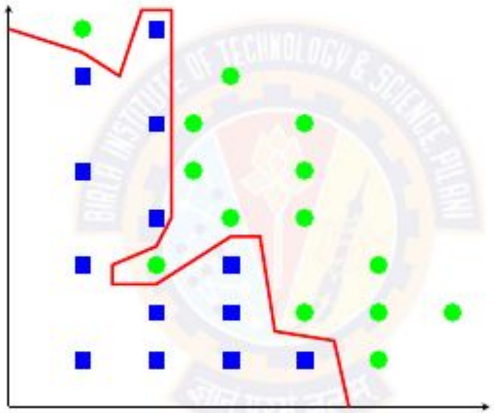
- The phenomenon of fitting the training data more closely than fit the underlying distribution is called **overfitting.**
- High Training accuracy and Low Validation accuracy
- Training error is significantly lower than the validation error.
- The techniques used to combat overfitting are called **regularization**.

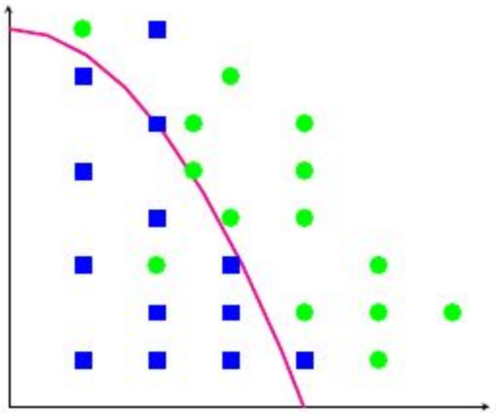
```
Training accuracy =  1.0
Validation accuracy =  0.4368932047515239З
Testing accuracy =  0.5614035087719298
```

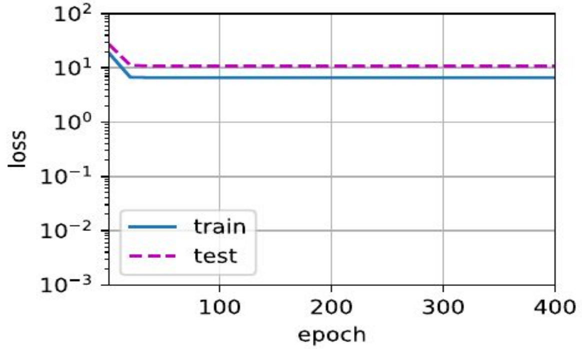


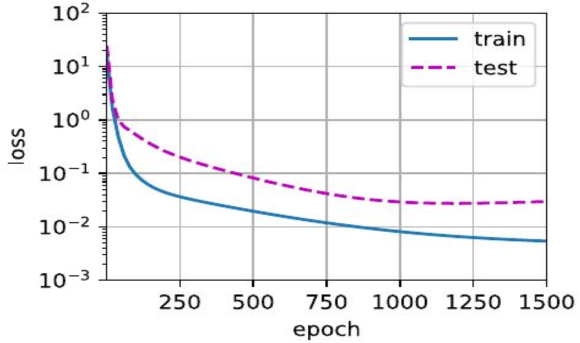
Underfitting or Overfitting?

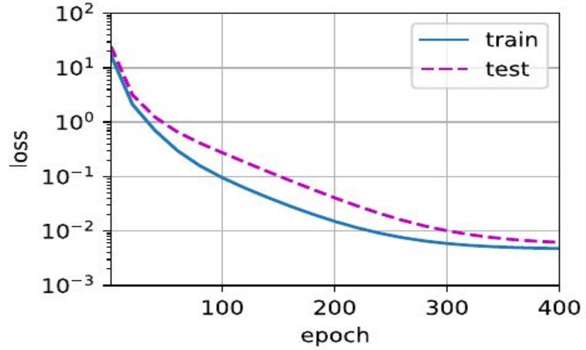

Simple Model
Underfitting


Complex Model
Overfitting

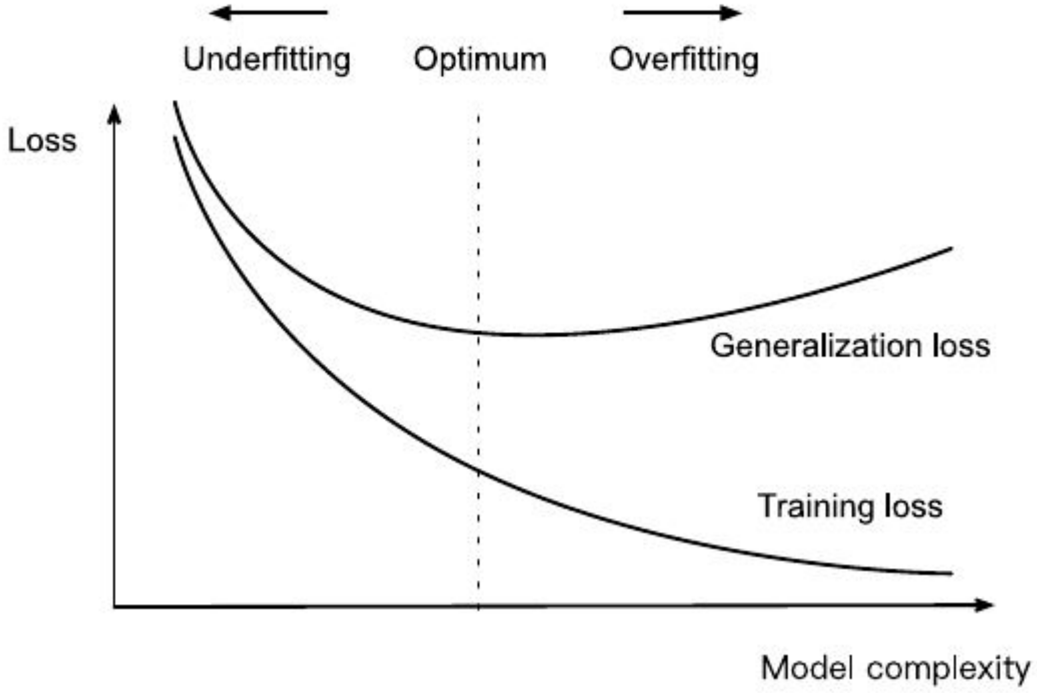

Just right model







Polynomial degree and underfitting vs. overfitting



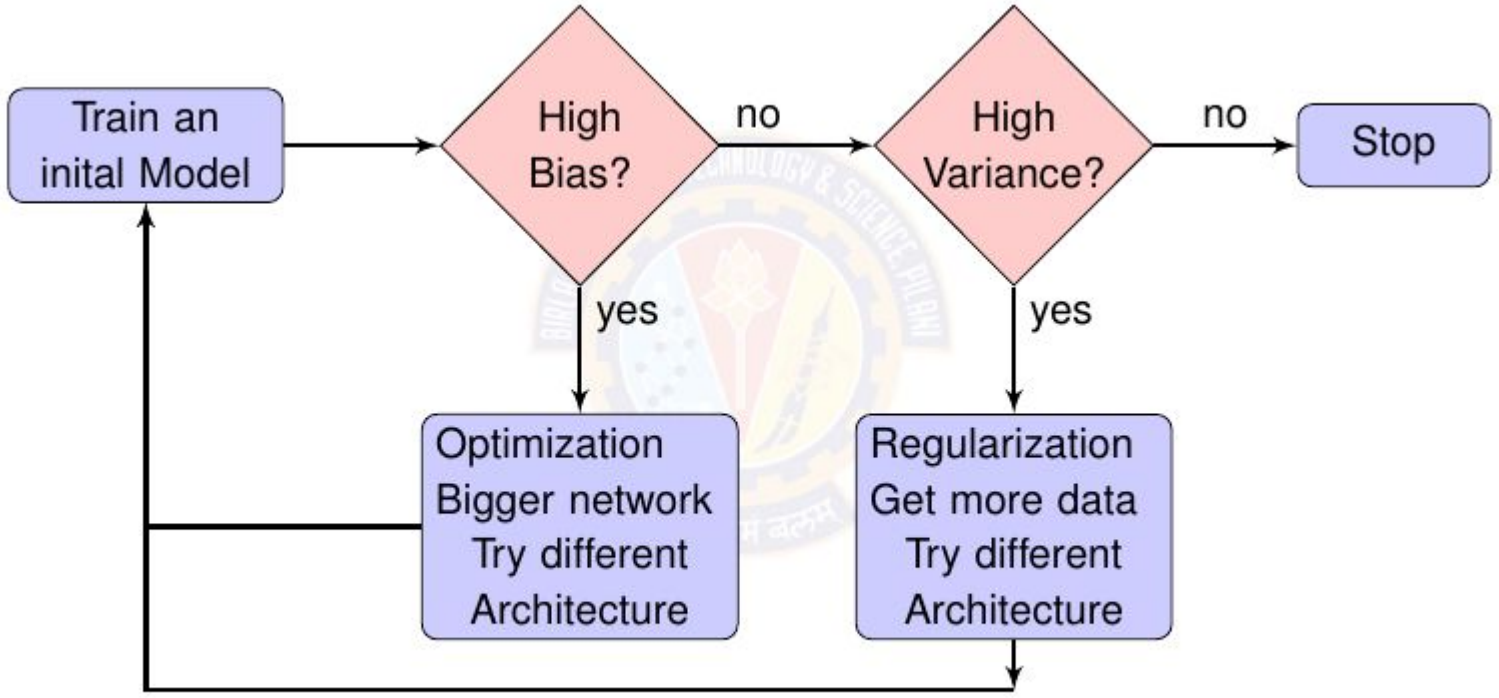
Model complexity and dataset size

- More data, fit a more complex model.
- More data, the generalization error typically decreases.

- Less data, simpler models may be more difficult to beat.
- Less data, more likely and more severely, models may over-fit.

The current success of deep learning owes to the current abundance of massive datasets due to Internet companies, cheap storage, connected devices, and the broad digitization of the economy.

Deep Learning Model Selection



Credit: Andrew Ng

Regularization

Regularization Techniques

- Weight Decay (L2 regularization)
- Dropout
- Early Stopping

Weight Decay

L2 Regularization

- Measure the complexity of a linear function $f(x) = w^{\top}x$ by some norm of its weight vector, e.g., $// w //^2$.
- Add the norm as a penalty term to the problem of minimizing the loss. This will ensure that the weight vector is small.
- The objective function becomes **minimizing the sum of the prediction loss and the penalty term.**
- L2-regularized linear models constitute the ridge regression algorithm.

L2 Regularization

- The trade off between standard loss and the additive penalty is given by regularization constant λ, a non-negative hyperparameter.

$$Objective function = \mathcal{L}(w, b) + \frac{\lambda}{2} \mid\mid w \mid\mid^2$$

$$Weight updation \qquad w \leftarrow (1 - \eta\lambda)w - \eta g$$

- For λ = 0, we recover the original loss function.
- For λ > 0, we restrict the size of // w // .
- Smaller values of λ correspond to less constrained w, whereas larger values of λ constrain w more considerably.
- By squaring the L2 norm, we remove the square root, leaving the sum of squares of each component of the weight vector.
- The derivative of a quadratic function ensure that the 2 and 1/2 cancel out.

L1 Regularization

- L1-regularized linear regression is known as lasso regression.
- L1 penalties lead to models that concentrate weights on a small set of features by clearing the other weights to zero. This is called feature selection.

$$Objective function = \mathcal{L}(w,b) + \lambda \sum_i \mid w_i \mid$$

$$Weight updation \qquad w \leftarrow (1 - \eta\lambda)w - \eta g$$

L2 Regularization	L1 Regularization
Sum of square of weights	Sum of absolute value of weights
Learn complex data patterns	Generates simple and interpretable models
Estimate mean of data	Estimate median of data
Not robust to outliers	Robust to outliers
Shrink coefficients equally	Shrink coefficients to zero
Non sparse solution	Sparse solution
One solution	Multiple solutions
No feature selection	Selects features
Useful for collinear features	Useful for dimensionality reduction

Dropout

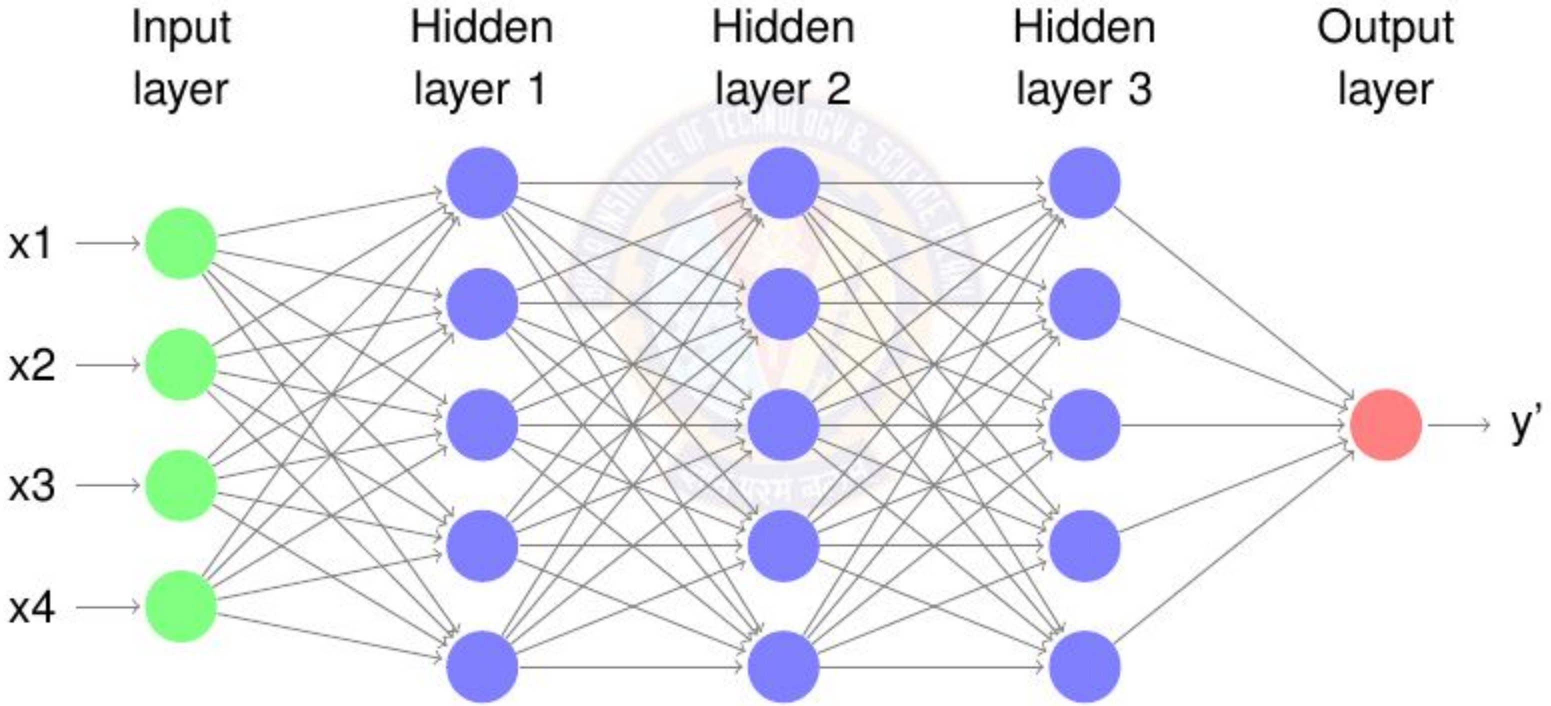
Smoothness

- Classical generalization theory suggests that to close the gap between train and test performance, aim for a simple model.
- Simplicity can be achieved
    - Using weight decay
    - Smoothness, i.e., that the function should not be sensitive to small changes to its inputs.
- Injecting noise enforces smoothness
    - training with input noise
    - inject noise into each layer of the network before calculating the subsequent layer during training.

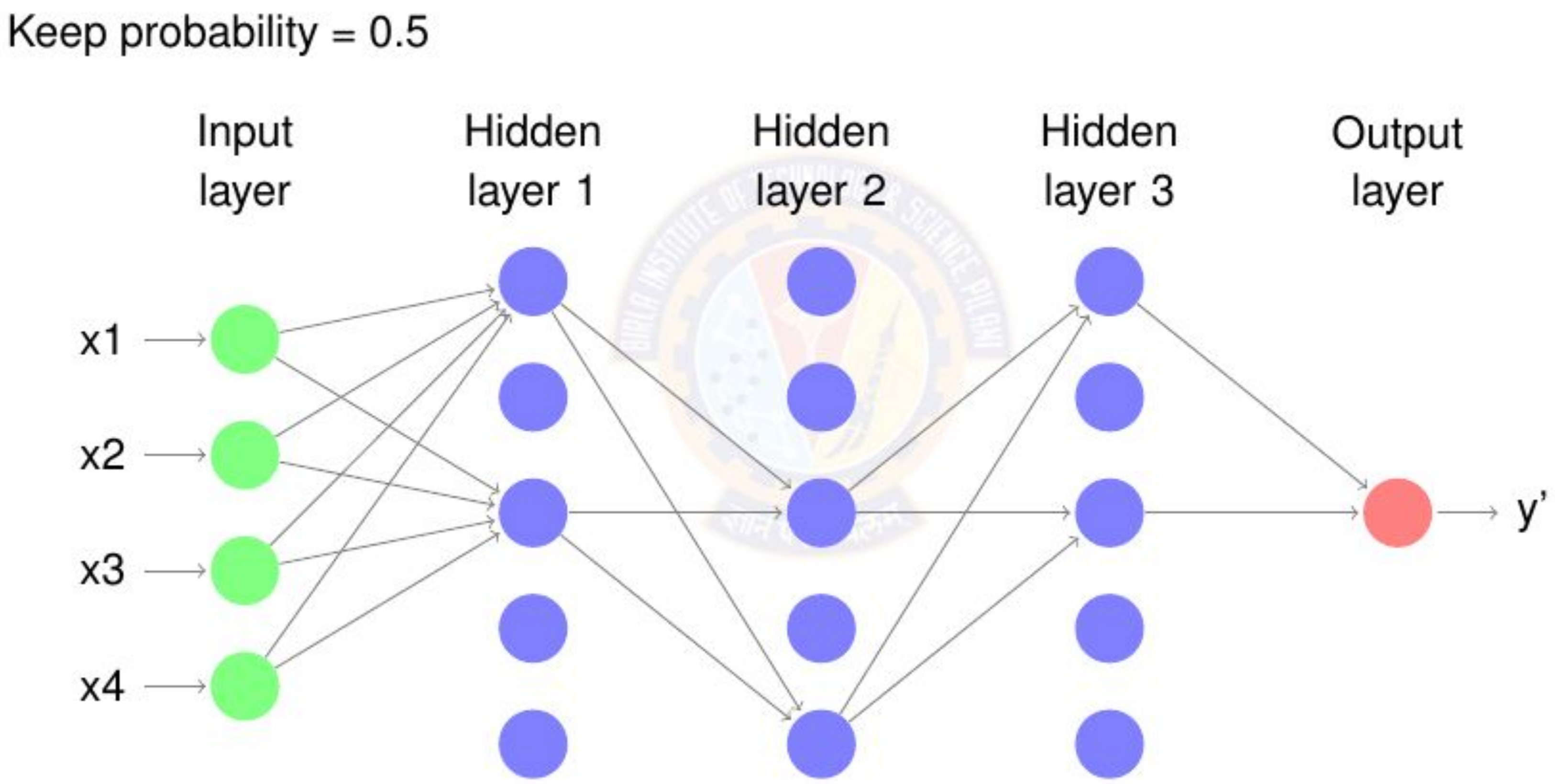
Dropout

- **Dropout** involves injecting noise while computing each internal layer during forward propagation.
- It has become a **standard** technique for training neural networks.
- The method is called **dropout** because we literally drop out some neurons during training.
- Apply dropout to a hidden layer, zeroing out each hidden unit with probability *p*.
- The calculation of the outputs no longer depends on dropped out neurons and their respective gradient also vanishes when performing backpropagation. (in that iteration)
- Dropout is disabled at test time.

Without Dropout

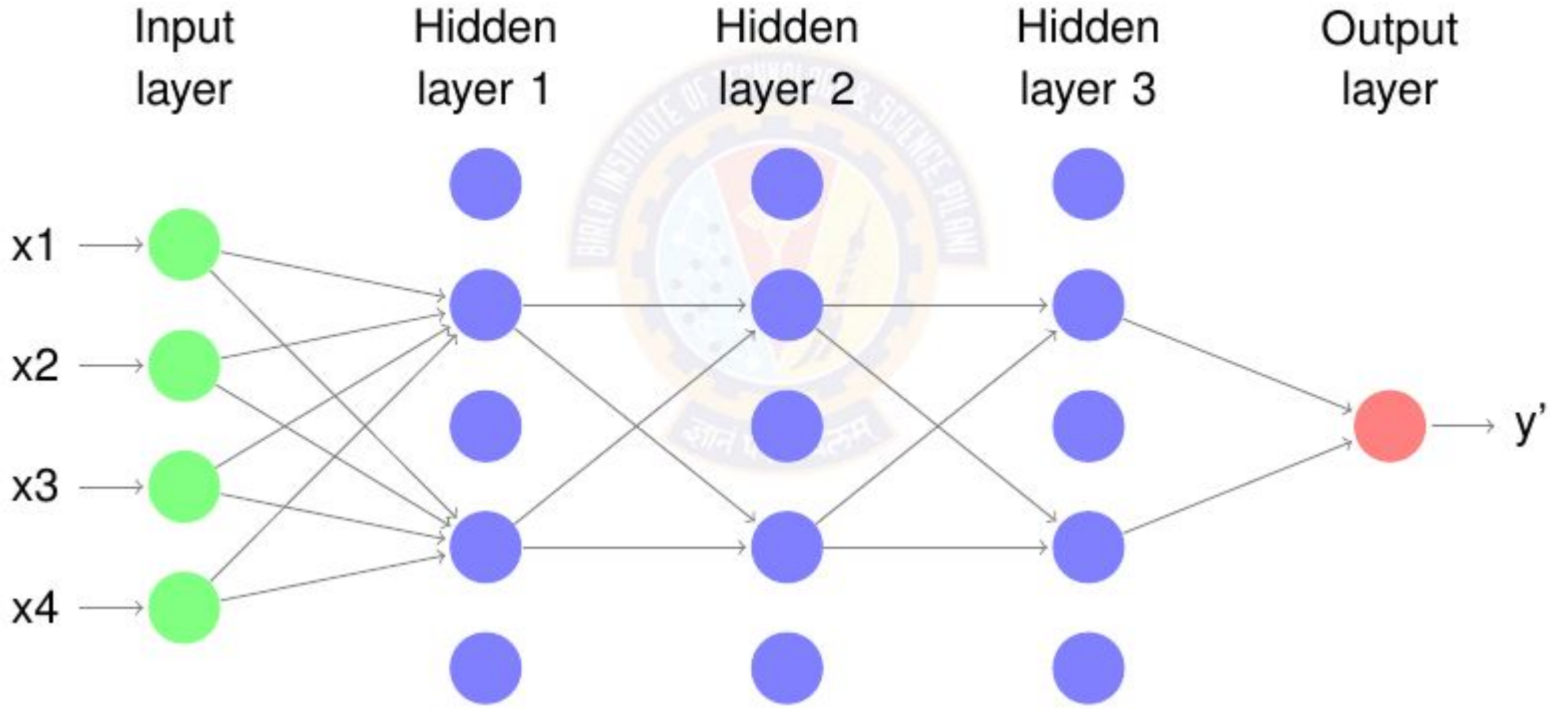


Dropout for first iteration



Dropout for second iteration

Keep probability = 0.5



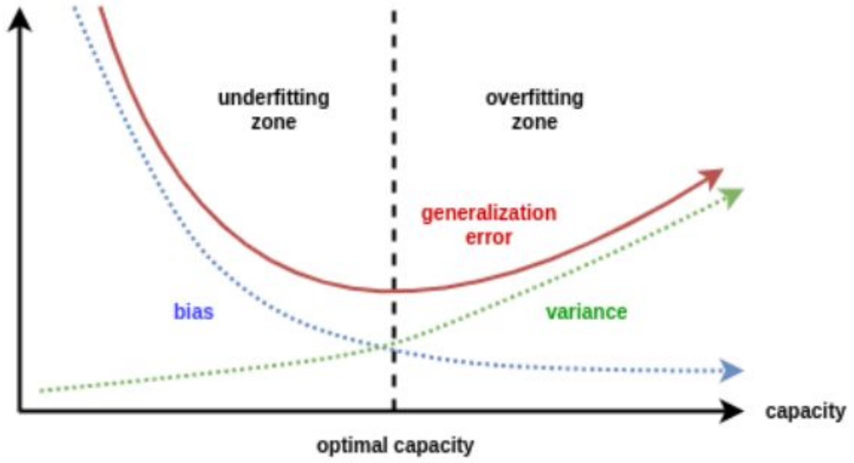
Dropout

- Dropout gives a smaller neural network, giving the effect of regularization.
- In general,
  - Vary keep probability (0.5 to 0.8) for each hidden layer.
  - The input layer has a keep probability of 1.0 or 0.9.
  - The output layer has a keep probability of 1.0.
-

Early Stopping

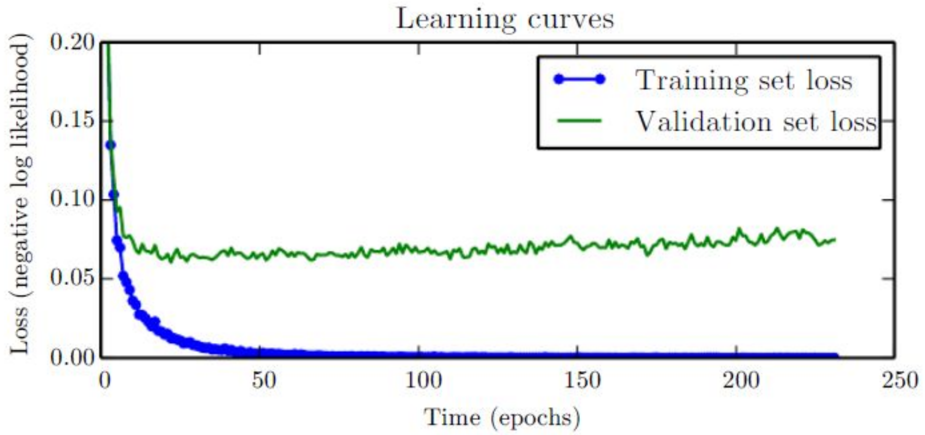
Before Early stopping

- When training large models, training error decreases steadily over time, but validation set error begins to rise again.
- Training objective decreases consistently over time.
- Validation set average loss begins to increase again, forming an asymmetric U-shaped curve.



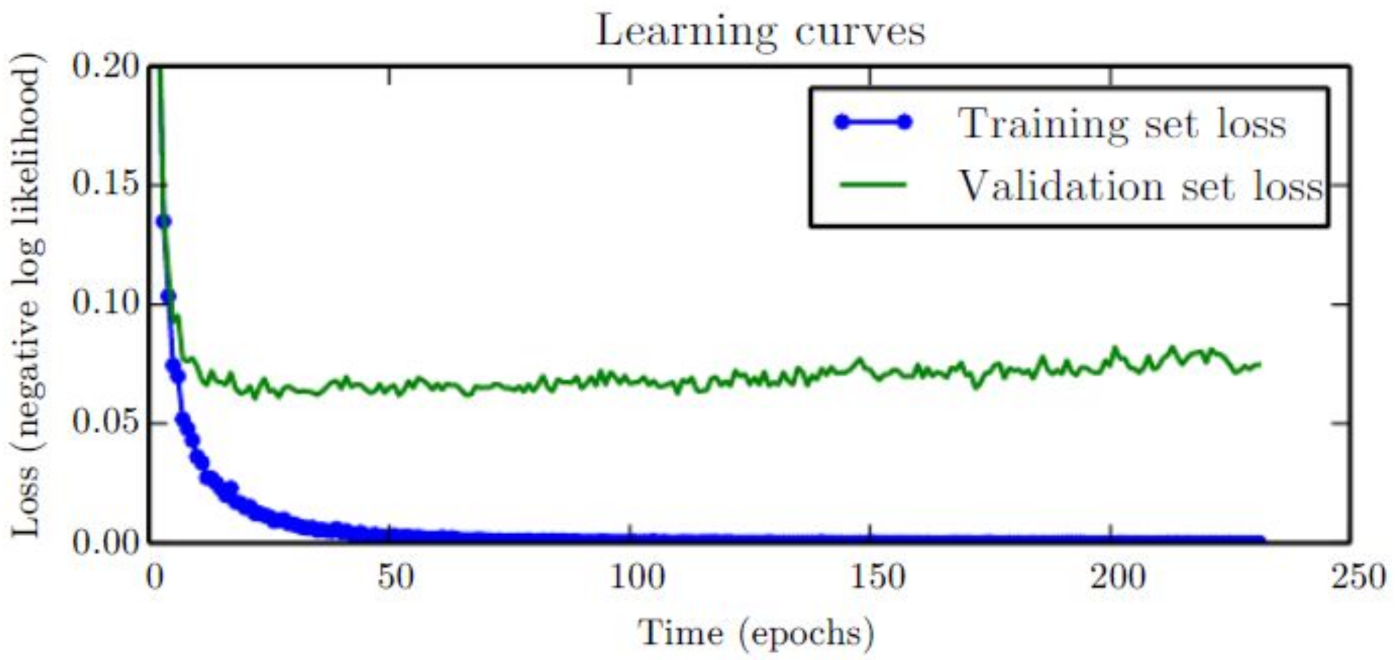
Early stopping

- No longer looking for a local minimum of validation error, while training.
- Train until the validation set error has not improved for some amount of time.
- Every time the error on the validation set improves, store a copy of the model parameters. When the training algorithm terminates, return these parar

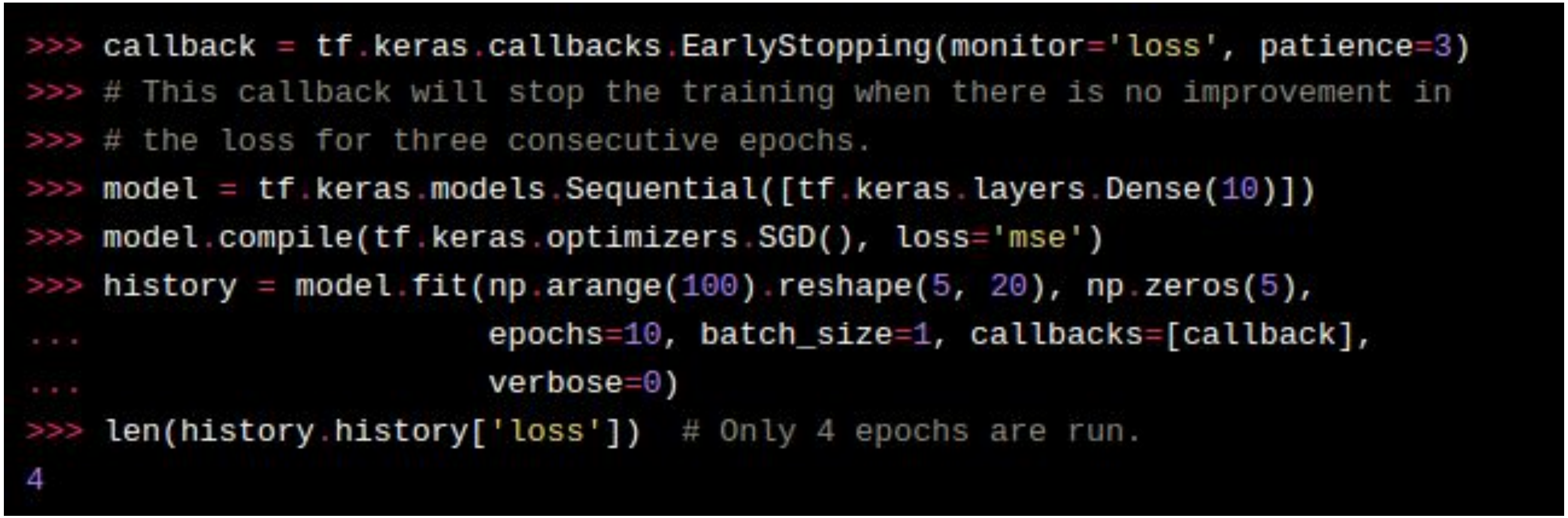


Early stopping

- Effective and simple form of regularization.
- Trains simpler models



Early Stopping code



```
>>> callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
>>> # This callback will stop the training when there is no improvement in
>>> # the loss for three consecutive epochs.
>>> model = tf.keras.models.Sequential([tf.keras.layers.Dense(10)])
>>> model.compile(tf.keras.optimizers.SGD(), loss='mse')
>>> history = model.fit(np.arange(100).reshape(5, 20), np.zeros(5),
...                     epochs=10, batch_size=1, callbacks=[callback],
...                     verbose=0)
>>> len(history.history['loss'])  # Only 4 epochs are run.
4
```

Numerical Stability and Initialization

Why Initialization is important?

- The choice of initialization is crucial for maintaining numerical stability.
- The choices of initialization can be tied up in interesting ways with the choice of the nonlinear activation function.
- Which function we choose and how we initialize parameters can determine how quickly our optimization algorithm converges.
- Poor choices can cause to encounter exploding or vanishing gradients while training.

Vanishing and Exploding Gradients

- Consider a deep network with **L** layers, input **x** and output **o**. With each layer **l** defined by a transformation **f$_l$** parameterized by weights **W(l)** , whose hidden variable is **h(l)**

$$\mathbf{h}^{(l)} = f_l(\mathbf{h}^{(l-1)}) \text{ and thus } \mathbf{o} = f_L \circ \ldots \circ f_1(\mathbf{x}).$$

- If all the hidden variables and the input are vectors, then the gradient of **o** with respect to any set of parameters **W(l)**

$$\partial_{\mathbf{W}^{(l)}} \mathbf{o} = \underbrace{\partial_{\mathbf{h}^{(L-1)}} \mathbf{h}^{(L)}}_{\mathbf{M}^{(L)} \stackrel{\text{def}}{=}} \cdot \ldots \cdot \underbrace{\partial_{\mathbf{h}^{(l)}} \mathbf{h}^{(l+1)}}_{\mathbf{M}^{(l+1)} \stackrel{\text{def}}{=}} \underbrace{\partial_{\mathbf{W}^{(l)}} \mathbf{h}^{(l)}}_{\mathbf{v}^{(l)} \stackrel{\text{def}}{=}}.$$

- Gradient is the product of **L − l** matrices M(L) · . . . · M(l+1) and the gradient vector v(l) .

Vanishing and Exploding Gradients

- The matrices M(l) may have a wide variety of eigenvalues. They might be small or large, and their product might be very large or very small.
- Gradients of unpredictable magnitude also threaten the stability of the optimization algorithms.
- Parameter updates may be either

  (i) excessively large, destroying our model (the **exploding gradient** problem);

  (ii) excessively small (the **vanishing gradient** problem), rendering learning impossible as parameters hardly move on each update.

Vanishing Gradients

- Activation function sigmoid σ can cause the vanishing gradient problem.
    - The sigmoid's gradient vanishes both when its inputs are large and when they are small.
    - When backpropagating through many layers, where the inputs to many of the sigmoids are close to zero, the gradients of the overall product may vanish.
- Solution: Use ReLU for hidden lakers. ReLU is more stable.

Parameter Initialization

1. Default Initialization
   - Used a normal distribution to initialize the values of the parameters.
2. Xavier Initialization
   - samples weights from a Gaussian distribution with zero mean and variance

$$' \sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}.$$

   - now-standard and practically beneficial

Batch Normalization

Why Batch Normalization?

1. Standardize the input features to each have a mean of zero and variance of one. This standardization puts the parameters a priori at a similar scale. Better optimization.
2. A MLP or CNN, as we train, the variables in intermediate layers may take values with widely varying magnitudes: both along the layers from the input to the output, across units in the same layer, and over time due to our updates to the model parameters. This drift in the distribution of such variables could hamper the convergence of the network.
3. Deeper networks are complex and easily capable of overfitting. This means that regularization becomes more critical.

Batch Normalization

- Batch normalization is a popular and effective technique that consistently accelerates the convergence of deep networks.
- Batch normalization is applied to individual layers.
- It works as follows:
    - In each training iteration, first normalize the inputs (of batch normalization) by subtracting their mean and dividing by their standard deviation, where both are estimated based on the statistics of the current minibatch.
    - Next, apply a scale coefficient and a scale offset.
- Due to the normalization based on batch statistics that batch normalization derives its name.
- Batch normalization works best for moderate minibatches sizes in the 50 to 100 range.

Batch Normalization

- Denote by **x** ∈ B an input to batch normalization (BN) that is from a minibatch B, batch normalization transforms **x** as

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}.$$

- $\hat{\mu}_B$ is the sample mean and $\hat{\sigma}_B$ is the sample standard deviation of the minibatch B.
- After applying standardization, the resulting minibatch has zero mean and unit variance.

Batch Normalization

- Denote by **x** ∈ B an input to batch normalization (BN) that is from a minibatch B, batch normalization transforms **x** as

$$\mathrm{BN}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}.$$

- $\hat{\mu}_B$ is the sample mean and $\hat{\sigma}_B$ is the sample standard deviation of the minibatch B.
- After applying standardization, the resulting minibatch has zero mean and unit variance.

Batch Normalization

- Elementwise scale parameter γ and shift parameter β that have the same shape as x. γ and β are parameters are learned jointly with the other model parameters.
- Batch normalization actively centers and rescales the inputs to each layer back to a given mean and size.
- Calculate $\hat{\mu}_B$ and $\hat{\sigma}_B$

$$\hat{\boldsymbol{\mu}}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x},$$

$$\hat{\boldsymbol{\sigma}}_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} (\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}})^2 + \epsilon.$$

Batch Normalization Layers

- Batch normalization implementations for fully-connected layers and convolutional layers are slightly different.
  - Fully-Connected Layers
    - Insert batch normalization after the affine transformation and before the nonlinear activation function.
  - Convolutional Layers
    - Apply batch normalization after the convolution and before the nonlinear activation function.
    - Carry out each batch normalization over the $m \cdot p \cdot q$ elements per output channel simultaneously.
- It operates on a full minibatch at a time.

Batch Normalization During Prediction

- After training, use the entire dataset to compute stable estimates of the variable statistics and then fix them at prediction time.

Ref TB Dive into Deep Learning

- Sections 5.4, 5.5, 5.6 and 8,5 (online version)

Next Session:
CNN