Intro to Neural Nets

Session 3: Model Fitting

Session Agenda

General Workflow

- Step 1: Get your model to overfit on training data (always possible).
- Step 2: Get your model to fit to validation data (this is quite exploratory)
- Step 3: Maximize out of sample performance by mitigating / delaying overfitting in training data.

Techniques to Mitigate Overfitting

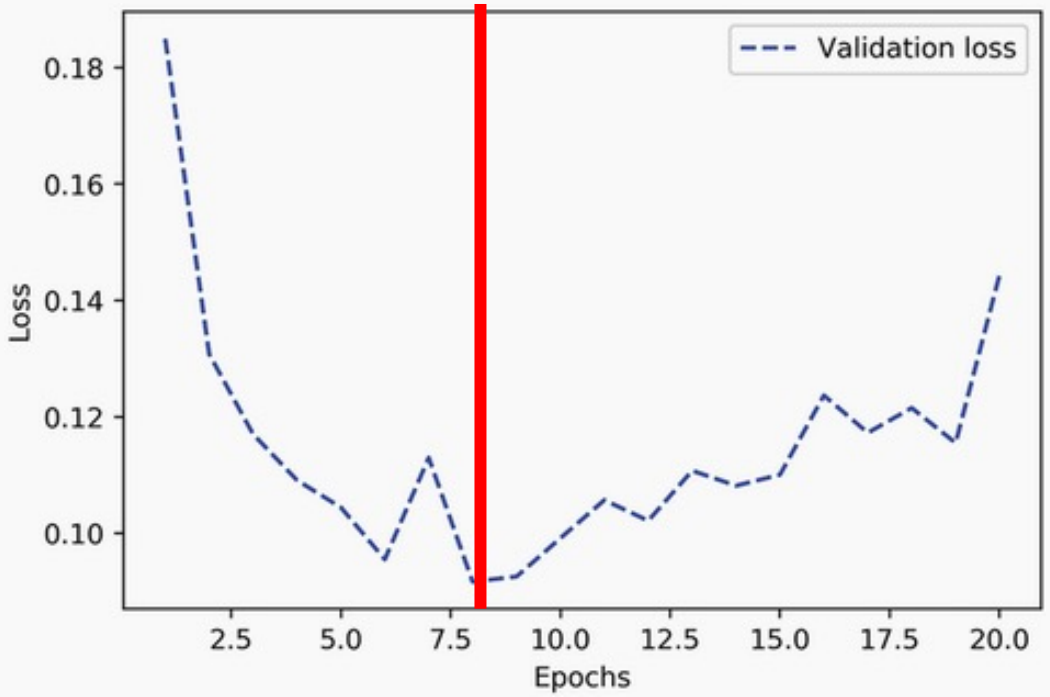
- Early stopping (we've seen this already, and we will do it regularly).
- Regularizing or constraining weights
- Dropout layer
- Adjust batch size
- Inject noise
- Get better / more data

Some Example Neural Nets

Early Stopping

Monitoring Validation Performance and then Manually Limiting Epoch Count

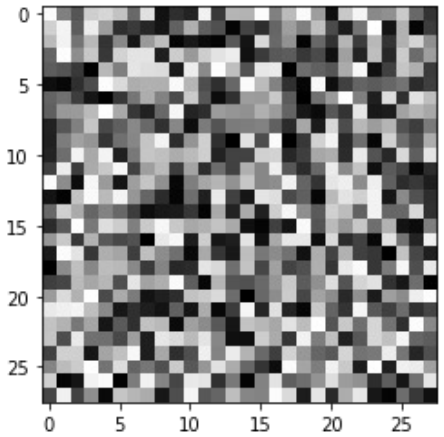
- This is the sole approach we have been taking thus far.

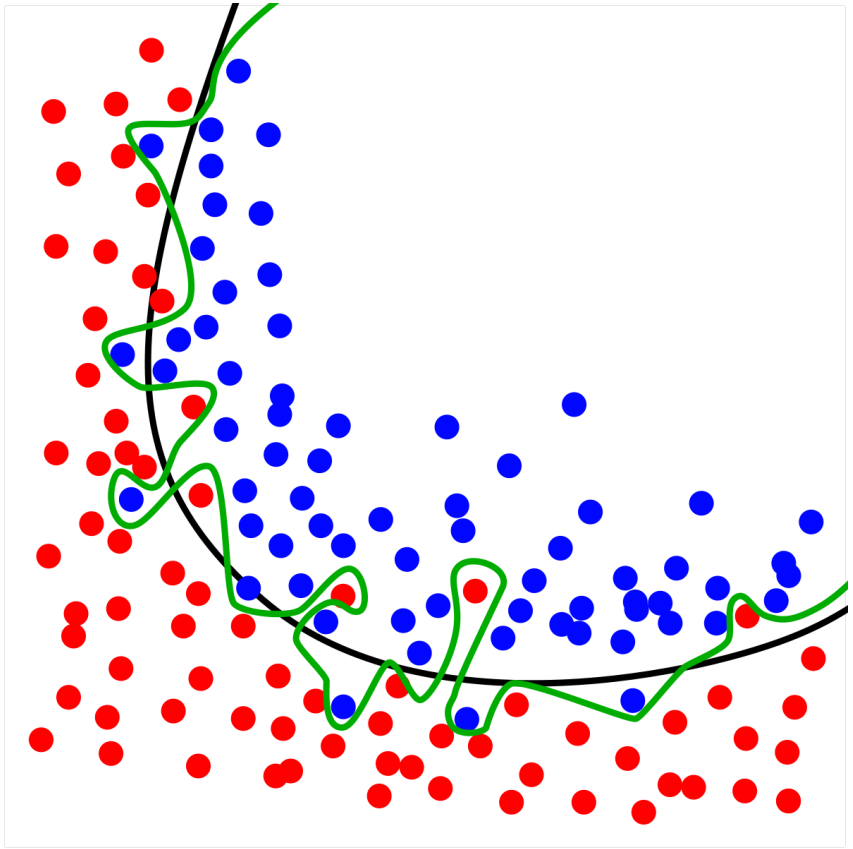


Overfitting

A Neural Network Can Easily Overfit…

- Let's look at an extreme case…





Regularizing Weights

Regularizing Weights Means Less Entropy

- Regularizing a layer's weights means the weights are updated less as their collective magnitude (e.g., sum) gets larger. Both L1 or L2 norms can be used here applied.
- This approach is typically used to improve the validation performance of smaller networks.

Listing 5.13 Adding L2 weight regularization to the model

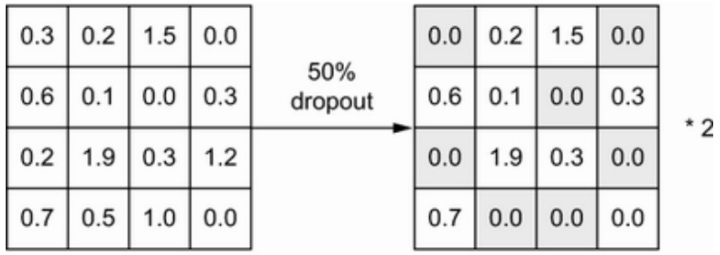
```
from tensorflow.keras import regularizers
model = keras.Sequential([
    layers.Dense(16,
                 kernel_regularizer=regularizers.l2(0.002),
                 activation="relu"),
    layers.Dense(16,
                 kernel_regularizer=regularizers.l2(0.002),
                 activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
history_l2_reg = model.fit(
    train_data, train_labels,
    epochs=20, batch_size=512, validation_split=0.4)
```

Dropout

Adding Dropout Layers to the Network

- Dropout layers set a random proportion of edge weights to 0 in a given training iteration. Typically between 20% and 50% of edges.
- When the final model is obtained, the 0's are removed, and the output values are scaled down uniformly to account for the change in the number of edges.
- This approach is more commonly used with large / deep networks.

Figure 5.20 Dropout applied to an activation matrix at training time, with rescaling happening during training. At test time the activation matrix is unchanged.



Adjust Batch Size

Deep Learning

Ian Goodfellow
Yoshua Bengio
Aaron Courville

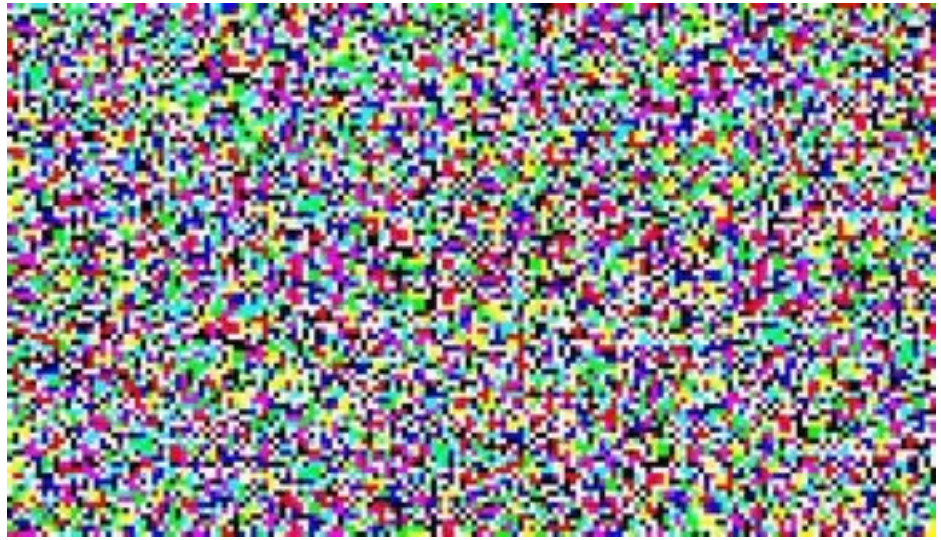
Minibatch sizes are generally driven by the following factors:

- Larger batches provide a more accurate estimate of the gradient, but with less than linear returns.
- Multicore architectures are usually underutilized by extremely small batches. This motivates using some absolute minimum batch size, below which there is no reduction in the time to process a minibatch.
- If all examples in the batch are to be processed in parallel (as is typically the case), then the amount of memory scales with the batch size. For many hardware setups this is the limiting factor in batch size.
- Some kinds of hardware achieve better runtime with specific sizes of arrays. Especially when using GPUs, it is common for power of 2 batch sizes to offer better runtime. Typical power of 2 batch sizes range from 32 to 256, with 16 sometimes being attempted for large models.
- Small batches can offer a regularizing effect (Wilson and Martinez, 2003), perhaps due to the noise they add to the learning process. Generalization error is often best for a batch size of 1. Training with such a small batch size might require a small learning rate to maintain stability because of the high variance in the estimate of the gradient. The total runtime can be very high as a result of the need to make more steps, both because of the reduced learning rate and because it takes more steps to observe the entire training set.

Inject Noise

Just Like Smaller Batch Size, but Purposeful

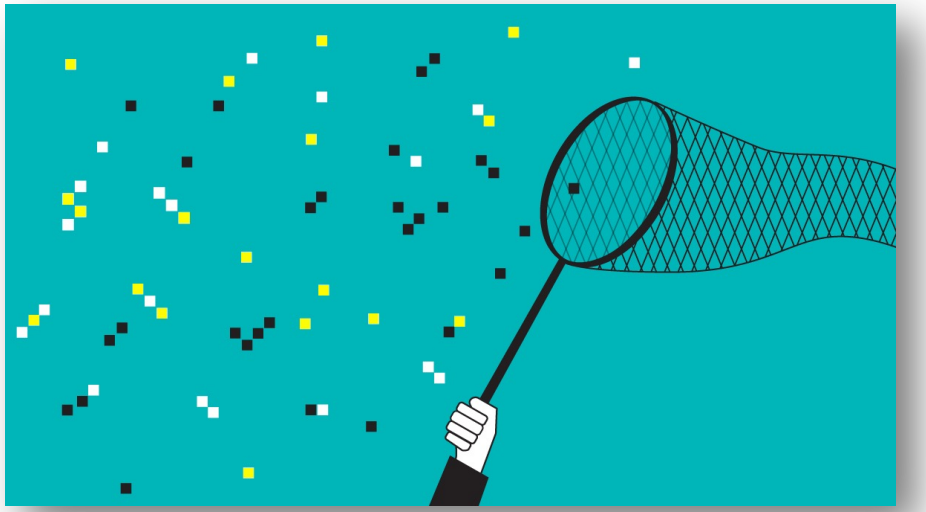
- One can manually jitter model weights at each iteration by ‘adding’ random noise.
- You can add noise to any model component, including the inputs, activations and outcome labels.



Get More Data!

More Training Examples

- Means you can have a bigger training data-set, which will presumably contain more information for the model to extract.
- This is the approach that often yields the best marginal returns, though it can also be most costly.



Some Rules of Thumb

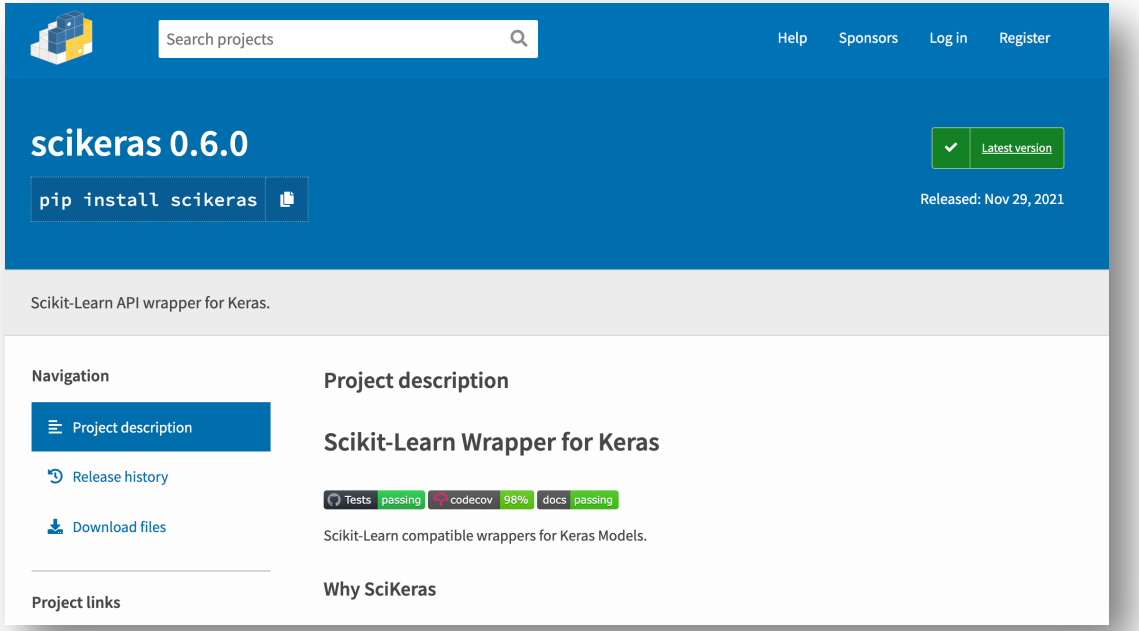
These Are Useful Guidelines for Your First Pass at a NN

- Start with 2 hidden layers.
- Give the first hidden layer (num_inputs / 2) nodes and the next (num_inputs / 4) nodes. If you add layers, decay the node count in this manner as you go along.
- Use ReLU (or SeLU) activations for hidden layers.
- Add Dropout at every layer after the input, with a rate of 0.5 (don't push beyond 0.5).
- If learning is flat, then more nodes in each layer.
- Whiten continuous input data (demean, divide by standard deviation).
- Make sure you are doing cross-validation, with a test set holdout.
- For classification problems, apply class weights to balance labels (in the model.fit() function).
- User RMSprop or Adam as your optimizer.
- Choose an appropriate loss function!
- Monitor accuracy as metric for classification problems, MSE for regression.
- Start with 20 epochs, increase if the validation loss has not yet reached its low point.
- Start with a batch size of 16 and then double it from there.

Scikit-Learn Wrapper for Keras

Facilitate Hyperparameter Tuning and Cross-validation of a Deep Net

- These wrappers can be used with the Sequential API, with two caveats:
- First, you need to formally specify the shape of the input layer.
- Second, you need to install scikeras in Google colab to use it.



Let's Walk Through a Couple Examples