Below is a distilled collection of conversations, messages, and debates I've had with peers and students on how to optimize deep models. If you have tricks you've found impactful, please share them!!

First, Why Tweak Models?

Deep learning models like the Convolutional Neural Network (CNN) have a massive number of parameters; we can actually call these hyper-parameters because they are not optimized inherently in the model. You could gridsearch the optimal values for these hyper-parameters, but you'll need a lot of hardware and time. So, does a true data scientist settle for guessing these essential parameters?

One of the best ways to improve your models is to build on the design and architecture of the experts who have done deep research in your domain, often with powerful hardware at their disposal. Graciously, they often open-source the resulting modeling architectures and rationale.

Deep Learning Techniques

Here are a few ways you can improve your fit time and accuracy with pre-trained models:

1. ***Research the ideal pre-trained architecture:*** Learn about the benefits of transfer learning, or browse some powerful CNN architectures. Consider domains that may not seem like obvious fits, but share potential latent features.

2. ***Use a smaller learning rate:*** Since pre-trained weights are usually better than randomly initialized weights, modify more delicately! Your choice here depends on the learning landscape and how well the pre-training went, but check errors across epochs for an idea of how close you are to convergence.

3. ***Play with dropout:*** As with Ridge and LASSO regularization for regression models, there is no optimized *alpha* or *dropout* for all models. It's a hyper-parameter that depends on your specific problem, and must be tested. Start with bigger changes — a wider gridsearch span across orders of magnitude,

like `np.logspace()` can provide— then drop down as with the learning rate above.

4. ***Limit weight sizes:*** We can limit the max norm (absolute value) of the weights for certain layers in order to generalize our model

5. ***Don't touch the first layers:*** The first hidden layers of a neural network tend to capture universal and interpretable features, like shapes, curves, or interactions that are very often relevant across domains. We should often leave these alone, and focus on optimizing the meta[2] latent level further back. This may mean adding hidden layers so we don't rush the process!

6. ***Modify the output layer:*** Replace model defaults with a new activation function and output size that is appropriate for your domain. However, don't limit yourself to the most obvious solution. While MNIST may seem like it wants 10 output classes, some numbers have common variations, and allowing for 12–16 classes may allow better settling of these variants and improved model performance! As with the tip above, deep learning models should be increasingly modified and tailored as we near output.

Techniques in Keras

Here's how to modify dropout and limit weight sizes in Keras with MNIST:

```
# dropout in input and hidden layers
# weight constraint imposed on hidden layers
# ensures the max norm of the weights does not exceed 5

model = Sequential()

model.add(Dropout(0.2, input_shape=(784,))) # dropout on the
inputs
# this helps mimic noise or missing data

model.add(Dense(128, input_dim=784, kernel_initializer='normal',
activation='relu', kernel_constraint=maxnorm(5)))

model.add(Dropout(0.5))

model.add(Dense(128, kernel_initializer='normal',
activation='tanh', kernel_constraint=maxnorm(5)))

model.add(Dropout(0.5))
```

```
model.add(Dense(1, kernel_initializer='normal',
activation='sigmoid'))
```

Dropout Best Practices:

- Use small dropouts of 20–50%, with 20% recommended for inputs. Too low and you have negligible effects; too high and you underfit.
- Use dropout on the input layer as well as hidden layers. This has been proven to improve deep learning performance.
- Use a large learning rate with decay, and large momentum.
- Constrain your weights! A big learning rate can result in exploding gradients. Imposing a constraint on network weight — such as max-norm regularization with a size of 5 — has been shown to improve results.
- Use a larger network. You are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations.

Here's an example of final layer modification in Keras with 14 classes for MNIST:

```
from keras.layers.core import Activation, Dense

model.layers.pop() # defaults to last
model.outputs = [model.layers[-1].output]
model.layers[-1].outbound_nodes = []
model.add(Dense(14, activation='softmax'))
```

And an example of how to freeze weights in the first five layers:

```
for layer in model.layers[:5]:
    layer.trainable = False
```

Alternatively, we can set the learning rate to zero for that layer, or use per-parameter adaptive learning algorithm like Adadelta or Adam. This is somewhat complicated and better implemented in other platforms, like Caffe.

Galleries of Pre-trained Networks:

Keras

- Kaggle List
- Keras Application
- OpenCV Example

TensorFlow

- VGG16
- Inception V3
- ResNet

Torch

- LoadCaffe

Caffe

- Model Zoo

View your TensorBoard graph within Jupyter

It's often essential to get a visual idea of how your model looks. If you're working in Keras, abstraction is nice but doesn't allow you to drill down into sections of your model for deeper analysis. Fortunately, the code below lets us visualize our models directly with Python:

```
# From:
http://nbviewer.jupyter.org/github/tensorflow/tensorflow/blob/mast
er/tensorflow/examples/tutorials/deepdream/deepdream.ipynb
# Helper functions for TF Graph visualization
from IPython.display import clear_output, Image, display, HTML
def strip_consts(graph_def, max_const_size=32):
    """Strip large constant values from graph_def."""
    strip_def = tf.GraphDef()
    for n0 in graph_def.node:
        n = strip_def.node.add()
        n.MergeFrom(n0)
        if n.op == 'Const':
            tensor = n.attr['value'].tensor
            size = len(tensor.tensor_content)
            if size > max_const_size:
```

```
                tensor.tensor_content = bytes("<stripped %d
bytes>"%size, 'utf-8')
    return strip_def

def rename_nodes(graph_def, rename_func):
    res_def = tf.GraphDef()
    for n0 in graph_def.node:
        n = res_def.node.add()
        n.MergeFrom(n0)
        n.name = rename_func(n.name)
        for i, s in enumerate(n.input):
            n.input[i] = rename_func(s) if s[0]!='^' else
'^'+rename_func(s[1:])
    return res_def

def show_graph(graph_def, max_const_size=32):
    """Visualize TensorFlow graph."""
    if hasattr(graph_def, 'as_graph_def'):
        graph_def = graph_def.as_graph_def()
    strip_def = strip_consts(graph_def,
max_const_size=max_const_size)
    code = """
        <script>
          function load() {{
            document.getElementById("{id}").pbtxt = {data};
          }}
        </script>
        <link rel="import"
href="https://tensorboard.appspot.com/tf-graph-basic.build.html"
onload=load()>
        <div style="height:600px">
          <tf-graph-basic id="{id}"></tf-graph-basic>
        </div>
    """.format(data=repr(str(strip_def)),
id='graph'+str(np.random.rand()))

    iframe = """
        <iframe seamless style="width:800px;height:620px;border:0"
srcdoc="{}"></iframe>
    """.format(code.replace('"', '&quot;'))
    display(HTML(iframe))

# Visualizing the network graph. Be sure expand the "mixed" nodes
to see their
# internal structure. We are going to visualize "Conv2D" nodes.
graph_def = tf.get_default_graph().as_graph_def()
tmp_def = rename_nodes(graph_def, lambda
s:"/".join(s.split('_',1)))
show_graph(tmp_def)
```

Visualize your Model with Keras

This will plot a graph of the model and save it as a png file:

```
from keras.utils import plot_model
plot_model(model, to_file='model.png')
```

`plot` takes two optional arguments:

- `show_shapes` (defaults to False) controls whether output shapes are shown in the graph.
- `show_layer_names` (defaults to True) controls whether layer names are shown in the graph.

You can also directly obtain the `pydot.Graph` object and render it yourself, for example to show it in an ipython notebook :

```
from IPython.display import SVG
from keras.utils.visualize_util import model_to_dot

SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

I hope this collection helps with your machine learning projects! Please let me know how you optimize your deep learning models in the comments below, and connect with me on Twitter and LinkedIn!