

# ECE763 Computer Vision: Models, Learning and Inference (including Deep Learning)



HW1: out 01/17, due 01/31, [DONE]

HW2: out 02/06, due 02/19 [DONE]

· HW3: out 02/27, due 03/16 [DONE]

· ~~HW4: out 03/17, due 03/31 [cancelled]~~

· ~~HW5: out 04/01, due 04/15 [POSTED]~~

· Project1: out 02/04, due 02/27 [DONE]

· Project2: out 02/27, due 03/22 [DONE]

· **Project3: out 04/01, due 04/30 (no late days allowed)**

Instructor: Tianfu (Matt) Wu



# Instructions and Notes

- *How to submit your solutions:* put your report (word or pdf or powerpoint) and results images (.png) if had in a folder named [your\_unityid]\_proj03 (e.g., twu19\_proj03), and then compress it as a zip file (e.g., twu19\_proj03.zip). Submit the zip file through **moodle**.
- *If you miss the deadline and still have unused late days, please send your zip file to TAs and me (0.5-day counted if later <= 6 hours based on email timestamp).*
- **Important Note:** We will **NOT** accept any replacement of submission after deadline ([+late days you use]), even if you can show the time stamp of the replacement is earlier than the deadline. So, **please double-check whether you submit correct files.**
- **NOT LATE DAYS ALLOWED!**



# Instructions and Notes

- Project 3: 40%
  - Requirement:
    - final write-up (15%) and
    - self-contained reproducible code (25%): **if your code is not reproducible, you will lose 20%; So, please make sure what you report in the write-up are consistent with the code.**
  - Bonus points: 10
    - For novel ideas (including novel implementation of some existing techniques, e.g., in a significantly faster way under fair comparisons).
    - If you want to claim this, please add a **section 0** in your write-up to clearly present and justify **what's new**. **The novelty will be checked and judged by both TAs and the instructor.**

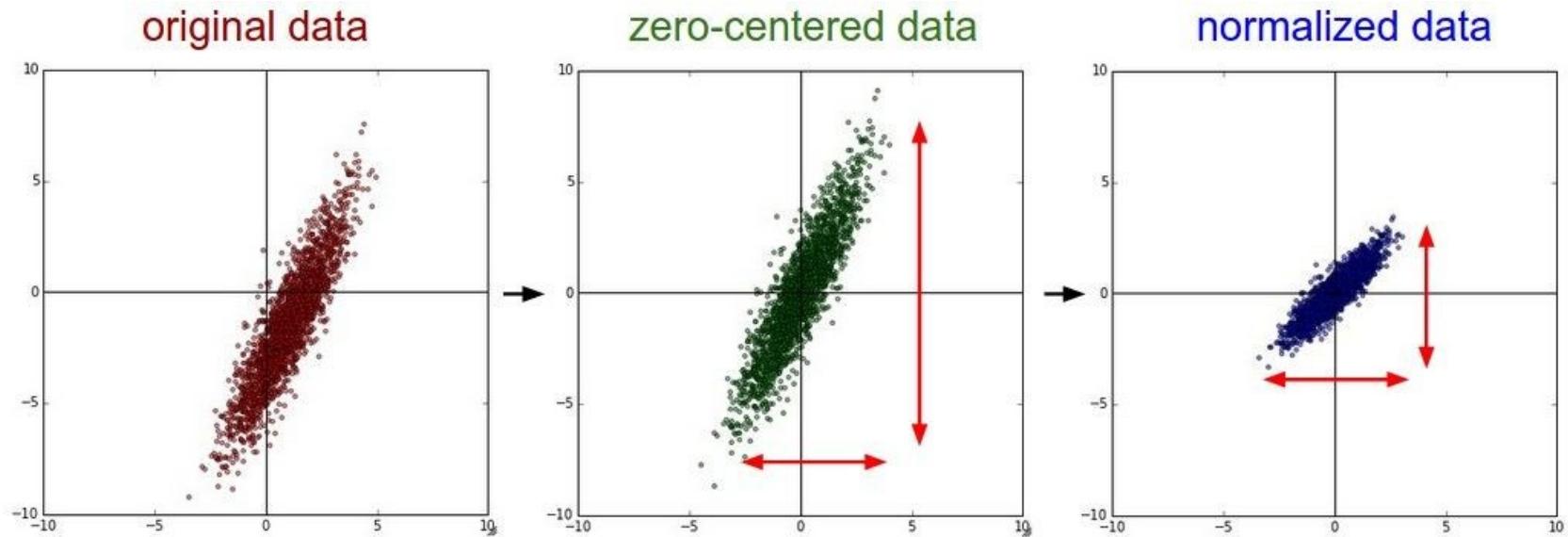


# Babysitting the training of DNN

- **Problem:** Practice the babysitting method of training neural network as discussed in Lecture note 21-22-23 (more specifically, slide 76-slide 94).
  - **Step 0:** Data (you can start with existing data provided in different DL code platform, e.g., MNIST in PyTorch), and you must test your code using the face data you used in project 01.
  - **Step 1:** preprocess the data and select a data augment scheme. (Note: you can compare w/ and w/o this preprocessing step and the data-aug to see how they affect your training and testing performance)
  - **Step 2:** Choose the architecture. E.g. you can use something simple, a 3-layer FC network or the LeNet 5 (available in all deep learning platforms).
  - > Then please follow the slides (see the next slides 4 - 23). to output detailed information of this babysitting procedure.
- **Hint:** You can reuse the tutorial code in different deep learning platform (e.g., the MNIST / CIFAR10 tutorial is available in almost all platforms).
  - E.g., If you use pytorch, you can find a lot of examples at <https://pytorch.org/tutorials/>
- **Requirement:** Although you can reuse the tutorial code, you need to modify the code to output different babysitting information (see the next slides 4 - 23). You need to snap your screenshots and paste those in your reports as shown in the slides. You also need to provide **self-contained reproducible code** .

# Babysitting the Learning Process

# Step 1: Preprocess the data and Select DataAug



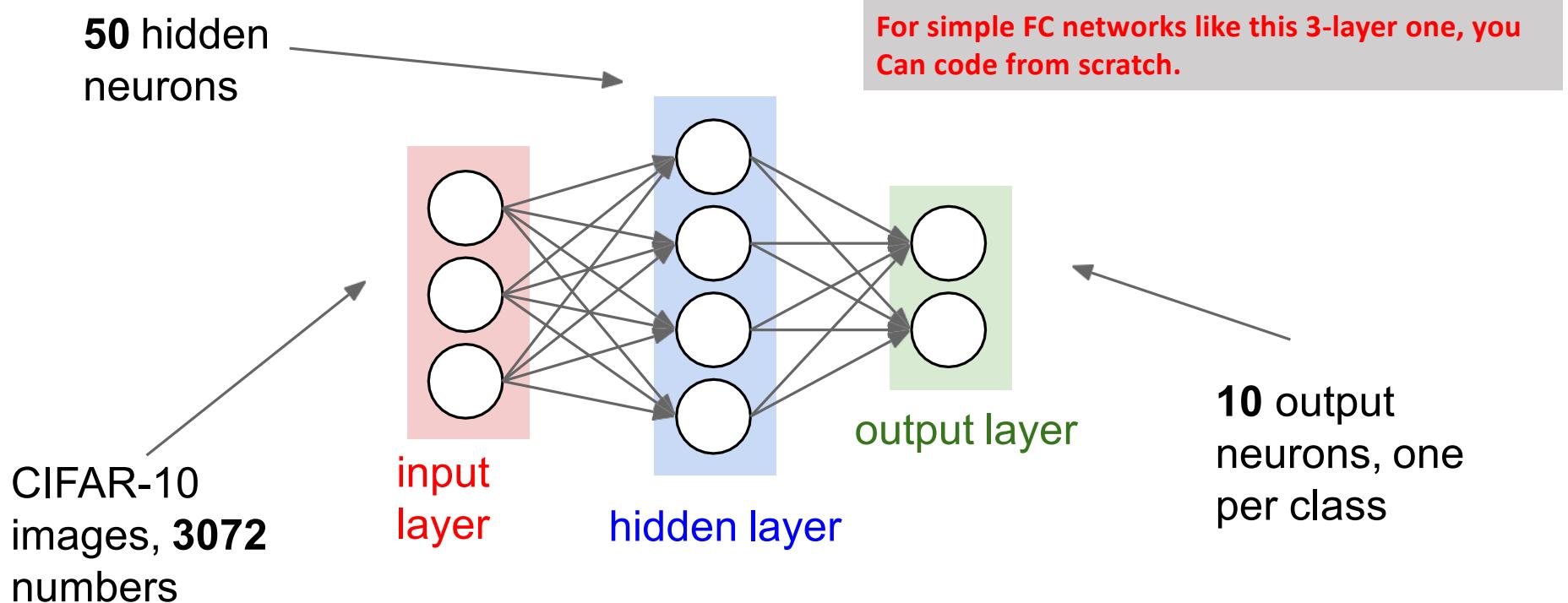
```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix,  
each example in a row)

You can try: with and w/o data normalization

Step 2: Choose the architecture:  
say we start with one hidden layer of 50 neurons:



# Double check that the loss is reasonable:

CHECK this step with your network and data

```
def init_two_layer_model(input_size, hidden_size, output_size):
    # initialize a model
    model = {}
    model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
    model['b1'] = np.zeros(hidden_size)
    model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
    model['b2'] = np.zeros(output_size)
    return model
```

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 0.0) disable regularization
print loss
```

2.30261216167

loss ~2.3.  
“correct” for  
10 classes

returns the loss and the  
gradient for all parameters

## Double check that the loss is reasonable:

```
def init_two_layer_model(input_size, hidden_size, output_size):
    # initialize a model
    model = {}
    model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
    model['b1'] = np.zeros(hidden_size)
    model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
    model['b2'] = np.zeros(output_size)
    return model
```

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 1e3)
```

crank up regularization

CHECK this step with your network and data

```
3.06859716482
```



loss went up, good. (sanity check)

## Lets try to train now...

**Tip:** Make sure that you can overfit very small portion of the training data

CHECK this step with your network and data

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

The above code:

take the first 20 examples from CIFAR-10

- turn off regularization (reg = 0.0)
- use simple vanilla ‘sgd’

Lets try to train now...

**Tip:** Make sure that you can overfit very small portion of the training data

Very small loss,  
train accuracy 1.00,  
nice!

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)

Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 4 / 200: cost 2.301196, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 5 / 200: cost 2.300044, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 6 / 200: cost 2.297864, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 7 / 200: cost 2.293595, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 8 / 200: cost 2.285096, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 9 / 200: cost 2.268094, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 10 / 200: cost 2.234787, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 11 / 200: cost 2.173187, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 12 / 200: cost 2.076862, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 13 / 200: cost 1.974090, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 14 / 200: cost 1.895885, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 15 / 200: cost 1.820876, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 16 / 200: cost 1.737430, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 17 / 200: cost 1.642356, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 18 / 200: cost 1.535239, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 19 / 200: cost 1.421527, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 20 / 200: cost 1.305750, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 195 / 200: cost 0.002694, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 196 / 200: cost 0.002674, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 197 / 200: cost 0.002655, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 198 / 200: cost 0.002635, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 199 / 200: cost 0.002617, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 200 / 200: cost 0.002597, train: 1.000000, val 1.000000, lr 1.000000e-03
finished optimization. best validation accuracy: 1.000000
```

# Lets try to train now...

Start with small regularization and find learning rate that makes the loss go down.

# Lets try to train now...

Start with small regularization and find learning rate that makes the loss go down.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                    model, two_layer_net,
                                    num_epochs=10, reg=0.000001,
                                    update='sgd', learning_rate_decay=1,
                                    sample_batches=True,
                                    learning_rate=1e-6, verbose=True)

Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing

# Lets try to train now...

Start with small regularization and find learning rate that makes the loss go down.

**loss not going down:**  
learning rate too low

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                    model, two_layer_net,
                                    num_epochs=10, reg=0.000001,
                                    update='sgd', learning_rate_decay=1,
                                    sample_batches=True,
                                    learning_rate=1e-6, verbose=True)

Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing: Learning rate is probably too low

# Lets try to train now...

Start with small regularization and find learning rate that makes the loss go down.

**loss not going down:**  
learning rate too low

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches=True,
                                  learning_rate=1e-6, verbose=True)

Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing: Learning rate is probably too low

Notice train/val accuracy goes to 20% though, what's up with that? (remember this is softmax)

## Lets try to train now...

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                   model, two_layer_net,
                                   num_epochs=10, reg=0.000001,
                                   update='sgd', learning_rate_decay=1,
                                   sample_batches = True,
```

- Start with small regularization and find learning rate that makes the loss go down.

Now let's try learning rate 1e6.

- loss not going down:
- learning rate too low

## Lets try to train now...

Start with small regularization and find learning rate that makes the loss go down.

**loss not going down:**  
learning rate too low  
**loss exploding:**  
learning rate too high

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                   model, two_layer_net,
                                   num_epochs=10, reg=0.000001,
                                   update='sgd', learning_rate_decay=1,
                                   sample_batches = True,
                                   learning_rate=1e6, verbose=True)

/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:50: RuntimeWarning: divide by zero en
countered in log
    data_loss = -np.sum(np.log(probs[range(N), y])) / N
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:48: RuntimeWarning: invalid value enc
ountered in subtract
    probs = np.exp(scores - np.max(scores, axis=1, keepdims=True))
Finished epoch 1 / 10: cost nan, train: 0.091000, val 0.087000, lr 1.000000e+06
Finished epoch 2 / 10: cost nan, train: 0.095000, val 0.087000, lr 1.000000e+06
Finished epoch 3 / 10: cost nan, train: 0.100000, val 0.087000, lr 1.000000e+06
```

cost: NaN almost always means high learning rate...

# Lets try to train now...

Start with small regularization and find learning rate that makes the loss go down.

**loss not going down:**  
learning rate too low  
**loss exploding:**  
learning rate too high

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=3e-3, verbose=True)

Finished epoch 1 / 10: cost 2.186654, train: 0.308000, val 0.306000, lr 3.000000e-03
Finished epoch 2 / 10: cost 2.176230, train: 0.330000, val 0.350000, lr 3.000000e-03
Finished epoch 3 / 10: cost 1.942257, train: 0.376000, val 0.352000, lr 3.000000e-03
Finished epoch 4 / 10: cost 1.827868, train: 0.329000, val 0.310000, lr 3.000000e-03
Finished epoch 5 / 10: cost inf, train: 0.128000, val 0.128000, lr 3.000000e-03
Finished epoch 6 / 10: cost inf, train: 0.144000, val 0.147000, lr 3.000000e-03
```

3e-3 is still too high. Cost explodes....

=> Rough range for learning rate we should be cross-validating is somewhere [1e-3 ... 1e-5]

# Hyperparameter Optimization

CHECK this step (weight decay, learning rate) with your network and data, as well as other hyperparameters which you observe to be sensitive in your training.

# Cross-validation strategy

**coarse -> fine** cross-validation in stages

**First stage:** only a few epochs to get rough idea of what params work

**Second stage:** longer running time, finer search

... (repeat as necessary)

Tip for detecting explosions in the solver:

If the cost is ever  $> 3 * \text{original cost}$ , break out early

# For example: run coarse search for 5 epochs

```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-5, 5)
    lr = 10**uniform(-3, -6) ←

    trainer = ClassifierTrainer()
    model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
    trainer = ClassifierTrainer()
    best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                              model, two_layer_net,
                                              num_epochs=5, reg=reg,
                                              update='momentum', learning_rate_decay=0.9,
                                              sample_batches = True, batch_size = 100,
                                              learning_rate=lr, verbose=False)
```

note it's best to optimize in log space!

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

nice

# Now run finer search...

```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-5, 5)
    lr = 10**uniform(-3, -6)
```

adjust range

```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-4, 0)
    lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good  
for a 2-layer neural net  
with 50 hidden neurons.

# Now run finer search...

```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-5, 5)
    lr = 10**uniform(-3, -6)
```

adjust range

```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-4, 0)
    lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100) ←
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good  
for a 2-layer neural net  
with 50 hidden neurons.

But this best  
cross-validation result is  
worrying. Why?

# Summary – Part 1

TLDRs

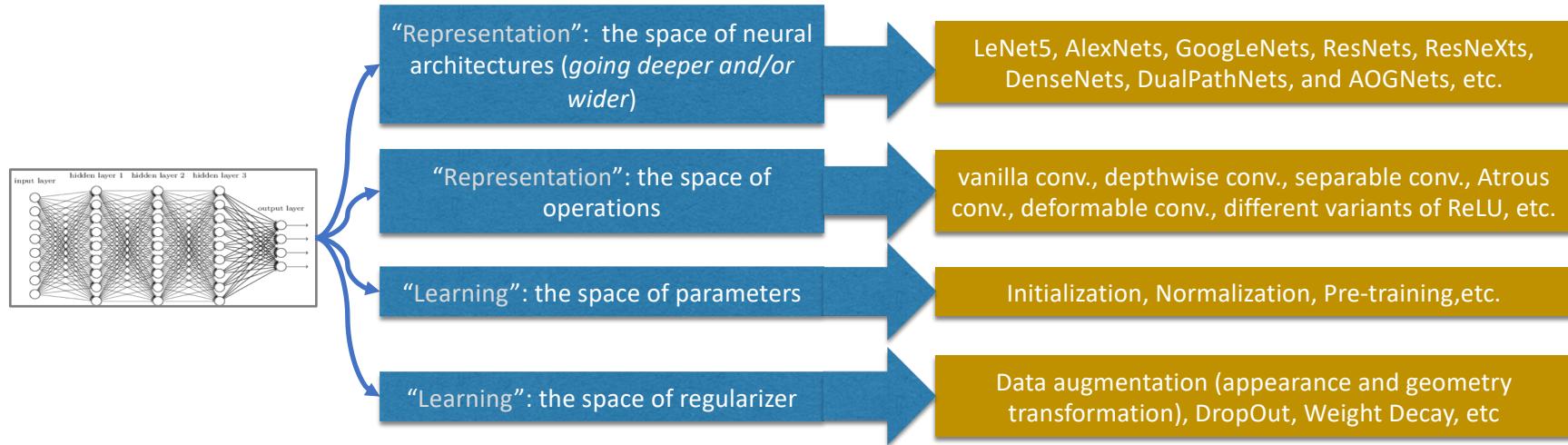
We looked in detail at:

- Activation Functions (use ReLU)
- Data Preprocessing (images: subtract mean)
- Weight Initialization (use Xavier/He init)
- Batch Normalization (use)
- Babysitting the training process



# Bonus Points for Creativity and Novelty

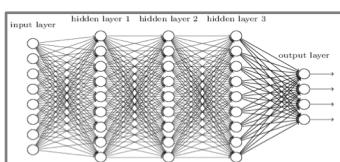
- Exploring new designs in, but not limited to, three aspects as follows.





# Bonus Points for Creativity and Novelty

- Potential Topic 1: New design of network architectures



“Representation”: the space of neural architectures (*going deeper and/or wider*)

LeNet5, AlexNets, GoogLeNets, ResNets, ResNeXts, DenseNets, DualPathNets, and AOGNets, etc.

- Understand existing neural architectures
- Be critical and creative to think about different designs
  - Exploring the space of DAGs for network topologies.
  - E.g., **exploiting your background domain knowledge**, if you work in the circuit design field, or if you are familiar with molecular structures in chemistry or fluid dynamics in physics or theoretical studies of DAGs in mathematics, you can leverage different principles of designing efficient and effective “information flows” in those fields. **[Be creative!]**
  - Some baselines: you can think about how to combine different existing designs together.
- Validate your ideas on CIFAR10 (and/or CIFAR100)
- If your ideas are very promising, I will try to help you test on ImageNet using a large GPU server.
  - If the results are good, we can submit to computer vision/AI conferences.



# Bonus Points for Creativity and Novelty

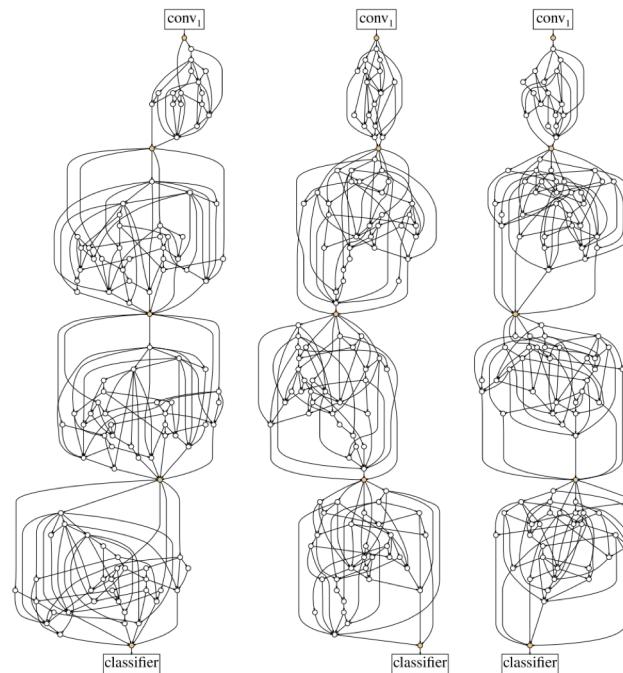
- Potential Topic 1: New design of network architectures
- Examples to inspire you

**Exploring Randomly Wired Neural Networks for Image Recognition**

Saining Xie Alexander Kirillov Ross Girshick Kaiming He

Facebook AI Research (FAIR)

Google and Read the paper





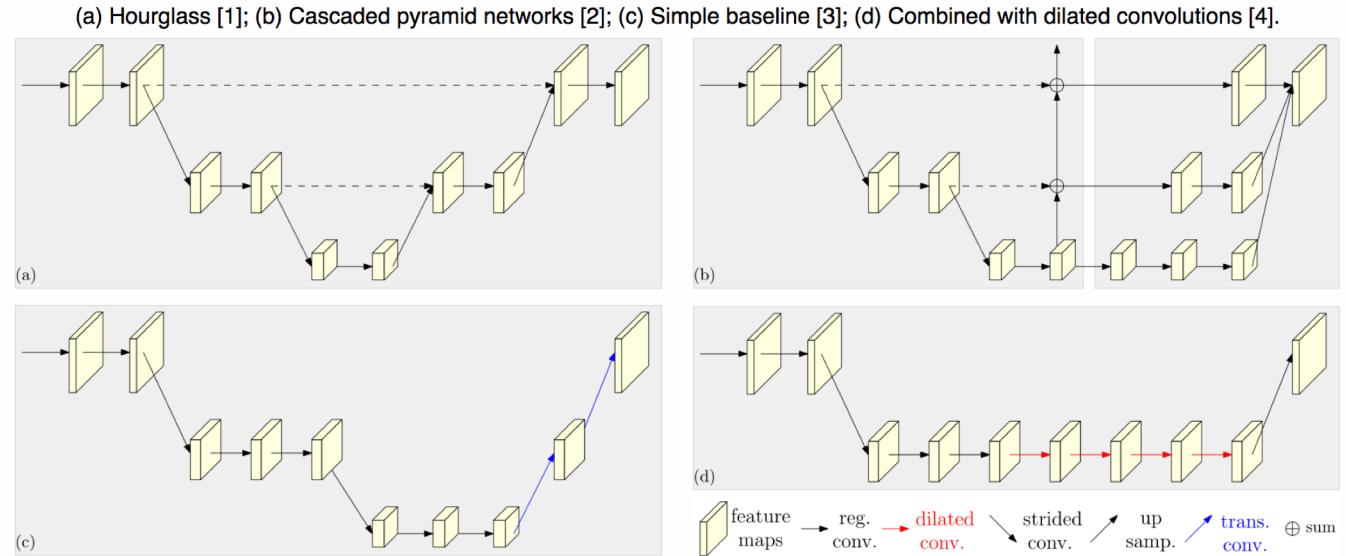
# Bonus Points for Creativity and Novelty

- Potential Topic 1: New design of network architectures
- Examples to inspire you

Deep High-Resolution Representation Learning for Human Pose Estimation

Ke Sun Bin Xiao Dong Liu Jingdong Wang

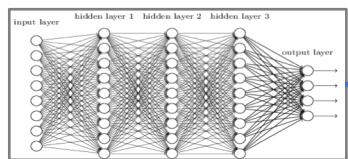
Google and Read the paper





# Bonus Points for Creativity and Novelty

- Potential Topic 2: New design of node operations



“Representation”: the space of operations

vanilla conv., depthwise conv., separable conv., Atrous conv., deformable conv., different variants of ReLU, etc.

- Understand existing operations
- Be critical and creative to think about different designs
  - Consider existing network architectures such as [ResNets](#) or [DenseNets](#) (PyTorch has default implementation), and how to extend their operations (e.g., they use the Bottleneck operation)
  - Exploring the combinations of different primitive operations using some micro-topologies similar in the spirit to network-in-network (<https://arxiv.org/abs/1312.4400>).
  - Examples (understand how they improved the existing networks):
    - SENet: <https://arxiv.org/abs/1709.01507>
    - Non-local networks: <https://arxiv.org/abs/1711.07971> <https://arxiv.org/abs/1811.11721>
    - Image Transformer: <https://arxiv.org/abs/1802.05751>
    - HetConv <https://arxiv.org/pdf/1903.04120.pdf>
  - Exploiting your background domain knowledge on methods of how to explore spatial and channel contextual information, or attention mechanism for operation design. **[Be creative!]**
- Validate your ideas on CIFAR10 (and/or CIFAR100)
- If your ideas are very promising, I will try to help you test on ImageNet or other big datasets using a large GPU server.
  - If the results are good, we can submit to computer vision/AI conferences.

# Bonus Points for Creativity and Novelty

- Potential Topic 2: New design of node operations
- Examples to inspire you

## Res2Net: A New Multi-scale Backbone Architecture

Shang-Hua Gao\*, Ming-Ming Cheng\*, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip Torr

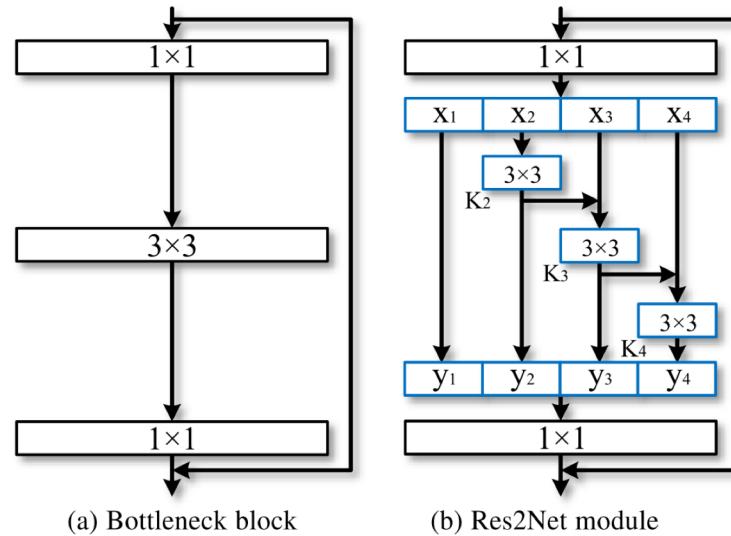


Fig. 2: Comparison between the bottleneck block and the proposed Res2Net module (the scale dimension  $s = 4$ ).

# Bonus Points for Creativity and Novelty

- Potential Topic 2: New design of node operations
- Examples to inspire you

## Drop an Octave: Reducing Spatial Redundancy in Convolutional Neural Networks with Octave Convolution

Yunpeng Chen<sup>†‡</sup>, Haoqi Fang<sup>†</sup>, Bing Xu<sup>†</sup>, Zhicheng Yan<sup>†</sup>, Yannis Kalantidis<sup>†</sup>, Marcus Rohrbach<sup>†</sup>, Shuicheng Yan<sup>‡§</sup>, Jiashi Feng<sup>†</sup>

<sup>†</sup>Facebook AI, <sup>‡</sup>National University of Singapore, <sup>§</sup>Qihoo 360 AI Institute

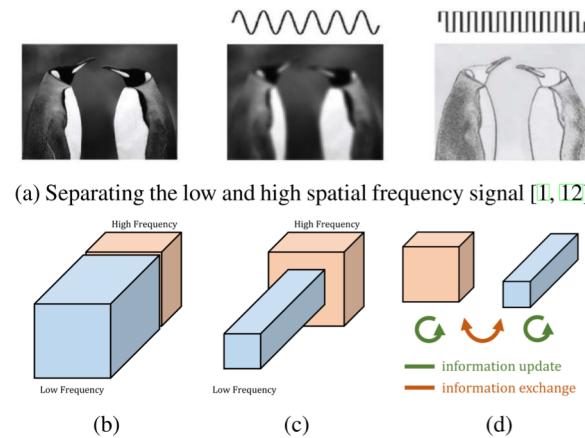
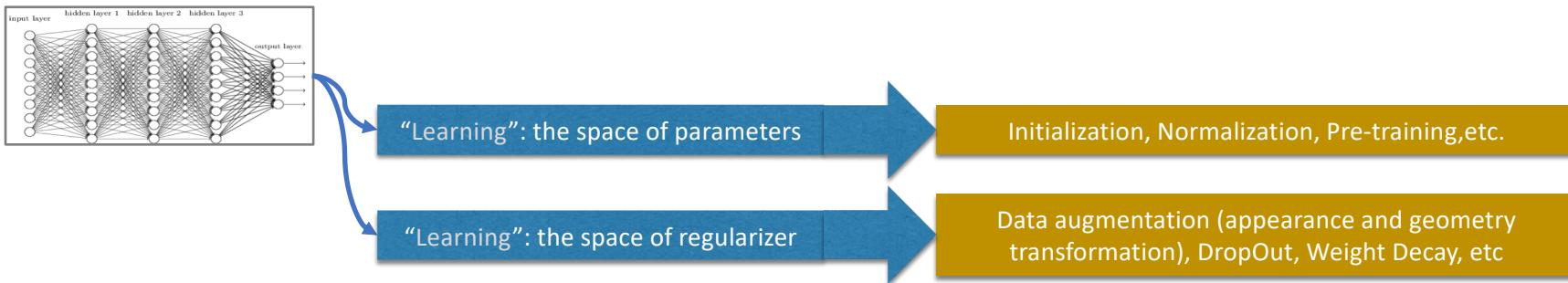


Figure 1: (a) Motivation. The spatial frequency model for vision [1, 12] shows that natural image can be decomposed into a low and a high spatial frequency part. (b) The output maps of a convolution layer can also be factorized and grouped by their spatial frequency. (c) The proposed multi-frequency feature representation stores the smoothly changing, low-frequency maps in a low-resolution tensor to reduce spatial redundancy. (d) The proposed Octave Convolution operates directly on this representation. It updates the information for each group and further enables information exchange between groups.



# Bonus Points for Creativity and Novelty

- Potential Topic 3: New design of learning

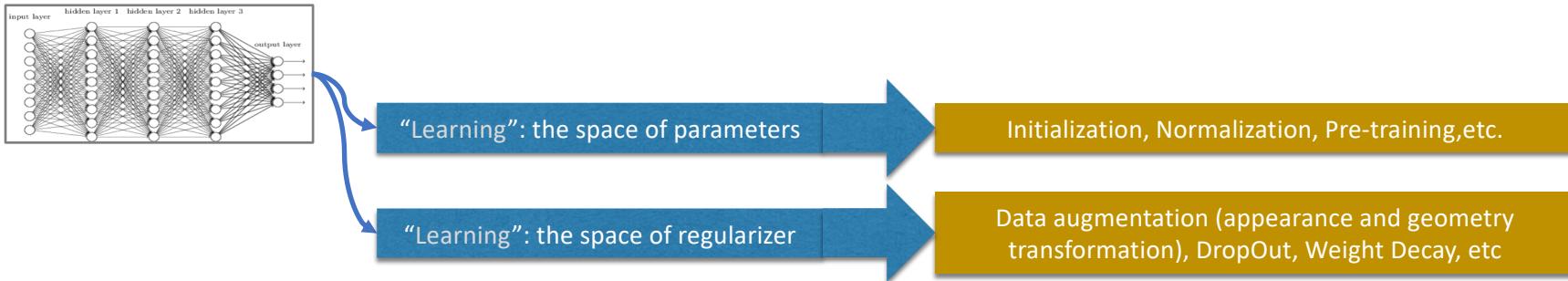


- Understand existing initialization and normalization methods
- Be critical and creative to think about different designs
  - Consider existing network architectures such as [ResNets](#) or [DenseNets](#) (PyTorch has default implementation), and how to train them with different initialization/normalization methods
  - Examples (understand how they improved the vanilla random initialization and the vanilla BatchNorm methods):
    - <https://arxiv.org/abs/1502.01852>
    - Fixup: <https://arxiv.org/abs/1901.09321v1>
    - BatchNorm, InstanceNorm, LayerNorm and GroupNorm: <https://arxiv.org/abs/1803.08494> (and related references therein)
    - How does BatchNorm work: <https://arxiv.org/abs/1805.11604>
    - SwitchableNorm: <https://github.com/switchablenorms/Switchable-Normalization>
    - Attentive Norm: <https://arxiv.org/abs/1908.01259>
    - Feature response norm: <https://arxiv.org/pdf/1911.09737.pdf>
- Validate your ideas on CIFAR10 (and/or CIFAR100)
- If your ideas very promising, I will try to help you test on ImageNet or other big datasets using a large GPU server.
  - If the results are good, we can submit to computer vision/AI conferences.



# Bonus Points for Creativity and Novelty

- Potential Topic 3: New design of learning



- Understand existing data augmentation "tricks"
- Be critical and creative to think about different designs
  - Consider existing network architectures such as [ResNets](#) or [DenseNets](#) (PyTorch has default implementation), and how to train them with different initialization/normalization methods
  - Examples (understand how they augment the training data):
    - Cutout: <https://arxiv.org/abs/1708.04552>
    - Mixup: <https://arxiv.org/abs/1710.09412>
    - Bag of tricks: <https://arxiv.org/abs/1812.01187> (and references for the tricks therein)
    - Grid Mask: <https://arxiv.org/abs/2001.04086>
  - Any transformation functions that do not change the task semantics are applicable. [Be creative!]
- Validate your ideas on CIFAR10 (and/or CIFAR100)
- If your ideas very promising, I will try to help you test on ImageNet or other big datasets using a large GPU server.
  - If the results are good, we can submit to computer vision/AI conferences.